TECHNOLOGY DEVELOPMENT AND CIRCUIT DESIGN FOR A PARALLEL LASER PROGRAMMABLE FLOATING POINT APPLICATION SPECIFIC PROCESSOI

THESIS

Michael W. Scriber
Captain, USAF

AFIT/GCE/ENG/89D-6

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY
Wright-Patterson Air Force Base, Ohio

AFIT/GCE/ENG/89D-6

TECHNOLOGY DEVELOPMENT AND CIRCUIT DESIGN FOR A PARALLEL LASER PROGRAMMABLE FLOATING POINT APPLICATION SPECIFIC PROCESSOR

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Engineering

Michael W. Scriber, B.S.C.E.

Captain, USAF

December 1989

Acknowledgments

I would like to thank my loving wife, Merrie', for her support and understanding. She was always there to listen in the times of frustration and rejoice in the times of success. I would also like to thank my children, Adam and Chelsea, for understanding when I had to say, "Daddy can't play right now. I have to study."

I would like to thank Captain Keith Jones, my thesis advisor and VLSI mentor, for patiently answering my endless barrage of questions. He supported me throughout my thesis effort, challenged me intellectually, and gave me the freedom to do it my way. I would like to thank Lieutenant Colonel Zdzislaw Lewantowicz and Captain Bruce George for their assistance in the parallel applications portion of this thesis and their time spent reading and critiquing my thesis. I would also like to thank Major Joseph DeGroat for his inspiration and guidance on the optimized carry multiplexed adder and the floating point adder architecture. He had a way of getting me fired up when I was in a slump.

I would like to thank the other students in the VLSI group for there ideas, support, and levity. I would especially like to thank Captain Steve "Bad Attitude" Pavick for enlightening me on the use of the DAS 9200, Captain Mike Dukes for spending an entire day trying to get the floating point adder files to MOSIS, Bruce Clay and Russ Milliron for helping me with the VLSI lab equipment, and Captain Doug Ford for his wire bonding expertise. I would also like to thank the researchers at Lincoln Laboratories for allowing me to use their laser equipment. I would especially like to thank Steve Gardner and Matthew Rhodes for their insight into diffusion linking and metal scribing.

My final thanks goes to God. He has blessed me with knowledge and insight far beyond my own capabilities. He has created opportunities for me to be challenged, to grow, and to succeed with Him. The successes of this thesis could not have been accomplished without the grace of God. In all things, give God the glory.

Michael W. Scriber

Table of Contents

List of Figures

List of Tables

Abstract

The laser programmable floating point application specific processor (LPASP) is a new approach at rapid development of custom VLSI chips. The LPASP is a generic application specific processor that can be programmed to perform a specific function. The effort of this thesis is to develop and test the double precision floating point adder and the laser programmable read-only memory (LPROM) that are macrocells within the LPASP. In addition, the thesis analyzes the applicability of an LPASP parallel processing system.

The double precision floating point adder is an adder/subtractor macrocell designed to comply with the IEEE double precision floating point standard. An 84-pin chip of the adder was fabricated using 2 micron feature sizes. The fastest processing time was measured at 120 nanoseconds over 23 worst case test vectors. The adder uses the optimized carry multiplexed (OCM) adder that was developed at AFIT.

The OCM adder is a new adder architecture that uses four parallel carry paths to attain a performance time on the order of $O(\sqrt[3]{n})$ with a gate count on the order of $O(n)$. The redundant logic associated with the parallel propagation banks is eliminated in the OCM adder so that the largest bit-slice of the adder contains only eight 2-to-1 multiplexer gates. A 57-bit adder was fabricated using 2 micron feature sizes. The processing time for the adder is 31 nsec.

The laser programmable read-only memory (LPROM) is programmed by using an argon-ion laser to cut transistor links. The LPROM was designed to provide a post-fabrication programmable capability to a MOSIS compatible ROM. A 256 by 16 bit LPROM was fabricated using 2 micron feature sizes. The chips were laser programmed with a laser programming yield of 100% and an off-chip read access time of 23 nanoseconds.

TECHNOLOGY DEVELOPMENT AND CIRCUIT DESIGN FOR A PARALLEL LASER PROGRAMMABLE FLOATING POINT APPLICATION SPECIFIC PROCESSOR

I. Introduction

The laser programmable application specific processor (LPASP) is a new approach at solving the computing problems that require very small and very fast computer processing. This chapter gives a broad introduction to the LPASP. It begins oy presenting some background as to why application specific processors are needed to solve today's computing problems. The chapter then explains the exact problem that the LPASP will be solving and what has been accomplished so far in the design of the chip. The scope of the research and the equipment requirements are also addressed.

1.1 Background

Today's computing problems for both aircraft and spacecraft require very fast computers for aerodynamic maneuvering, trajectory calculations, and object tracking. Most of our modern aircraft aerodynamic surfaces are controlled by computers. Without the computers on the F-16, for example, the aircraft is unstable. The on-board computers must be very fast to respond to aerodynamic instabilities and pilot controls. If the computers do not respond quickly to pilot controls, then the pilot tends to overcompensate the controls before the computer responds. An excellent example of pilot overcompensation is when the space shuttle made its first landing on Kennedy Space Center's runway. The computers didn't respond quickly to the pilots commands, so the pilot turned sharper. When the computers responded to the sharp turn, the resulting turn was too sharp and the shuttle nearly crashed.

Computers can be built that are very fast, but they usually are also very big. The size of a super-computer is too great to possibly fit on an aircraft or a satellite. The CRAY super-computers are so large that the outer housing contains cushions and is designed in the shape of a circular couch.

The solution to the problem of fast computing in a small package is to design application specific processor (ASP) chips that accomplish a specific task. The chips are very fast because they can be optimized to perform just one task.

1.2 Problem

Application specific processors satisfy the computing needs of today's modern aircraft and spacecraft, but since they only perform one task, a number of different ASPs must be designed to make a complete computer system. The task of designing the various ASPs to perform all of the operations required by an aircraft can be overwhelming. It takes approximately two years to conceive, develop, design, implement, and fabricate a custom non-silicon compiled ASP.

What is needed is a generic ASP that can be programmed to perform a specific function. The goal of this thesis research is to develop the technology and continue the circuit design for an ASP that can be modified by a laser to alter the ASPs configuration. By altering the configuration of the ASP, the generic ASP can be made to perform any number of different tasks. Since the ASP is laser programmable, it has been named the laser programmable application specific processor (LPASP).

The LPASP is capable of accomplishing a wide variety of tasks because it contains all of the common processing structures that are required by most computing algorithms. The LPASP includes within its architecture a double precision floating point adder/subtracter, a double precision floating point multiplier, two 32-bit integer arithmetic logic units (ALUs), a barrel shifter, two sets of 25 storage registers, six incrementers, and four address pointer/incrementer registers.

The programmability of the LPASP is achieved though the use of two read-only memories. The first ROM is called the fixed ROM because the information is fabricated into the ROM and cannot be changed. The fixed ROM contains common routines and functions that can be performed by the LPASP. The common routines include matrix multiplication, vector addition, and dot product calculations. The second ROM is the laser programmable read-only memory (LPROM). The LPROM is where the user specifies the exact task that the LPASP is to perform. The LPROM contains user specific routines and function calls to the routines that are stored in the fixed ROM.

Since the information in the LPROM is unique for each application, the data is not fabricated into the ROM like the fixed ROM's data. The data will be placed into the LPROM using a laser. The method for laser programming is one of the issues of this thesis effort.

1.3 Summary of Current Knowledge

The LPASP is the culmination of past thesis work accomplished by four individuals.The original work for the LPASP started with Captain David Gallagher's thesis effort to create a library of bit-slice standard cells that could be used to create an ASP(9). He created all of the

cells required for an integer ASP and culminated his thesis with the fabrication of a single-precision integer ASP.

The overall system architecture for the LPASP was designed by Captain John Comtois in 1988(4). He created all of the smaller structures including the ALUs, the register sets, the barrel shifter, and the control logic. Some of the structures either used Gallagher's standard cells or were derived from the standard cell library. He did not design the four major units, which are the double precision floating point multiplier, the double precision floating point adder/subtracter, the LPROM, and the fixed ROM.

The architecture for the double precision floating point multiplier was proposed by Captain Eric Fretheim as a class project(8). The multiplier combines octal Booth encoding with a parallel multiplier design. It receives two 64-bit numbers and returns the multiplication of the numbers as a 64-bit number. It is designed to operate in 80 nanoseconds. The multiplier is the largest single structure in the LPASP.

The basic cell design for the LPROM was attempted by Captain John Tillie(19). The cell design was similar to the fixed ROM except that the pulldown line was replaced by a diffusion link gap. In theory, when an argon ion laser beam strikes the diffusion link gap, the two pieces of diffusion weld together. Captain Tillie did not succeed in programming the memory cell because the precision equipment required for the task was not available. The diffusion link technology has been demonstrated very successfully at Lincoln Laboratories.

In 1985, Captain Paul Rossbach created a computer program that automatically designs an optimized ROM(12). The design uses cross-coupled transistors in the shape of an X, so it was called an XROM. The XROM program receives as input the data that needs to be stored in the fixed ROM. It then optimizes the ROM by swapping address lines and eliminating transistors. It outputs a mask level description of the fixed ROM design that is ready for fabrication. The data to be stored in the fixed ROM for the LPASP is being generated as part of Captain Bill Koch's thesis effort.

1.4 Scope

The original goal of this thesis effort was to complete the design and fabrication of the LPASP. Unfortunately, the diffusion link LPROM did not work, and the design of the floating point multiplier was not yet completed. Without the LPROM and the multiplier, the integration of the LPASP could not be performed. As a result, the goals of the thesis were modified. The first goal was to design, fabricate, and test the floating point adder. The second goal was to

design, fabricate, and test a new LPROM. The third goal was to analyze the feasibility for parallel applications of the LPASP.

1.5 Materials and Equipment

Once the LPROM chip is fabricated, it must be laser programmed at a laser programming workstation. The laser programming workstation consists of a laser with all of its peripheral cooling equipment, an optics box to guide the laser beam, an X-Y translation table to position the chip, and an isolation table to dampen vibrations from the floor. The positioning of the chip can be accomplished through pattern recognition of the program site in the LPROM. The pattern recognition equipment includes a Sun workstation to execute the software, a camera to capture the image, and a monitor to view the image.

The precision equipment required for laser programming costs approximately $250,000 and should be housed in a climate controlled room. Rather than purchase the equipment, it was found to be more cost effective to use the proven equipment at Lincoln Laboratories. The researchers at Lincoln Labs agreed to assist with the laser programming effort and were extremely helpful and generous.

1.6 Conclusion

The LPASP is a generic version of an application specific processor. It contains all of the capabilities and speed of a regular ASP, but it only requires about a days worth of laser programming time, compared to the two years needed to go from conception to fabrication for a custom ASP. The chip accomplishes the design speedup by containing all of the common processing structures and most of the common routines on a single chip that can be programmed with a laser to perform a specific task.

The complete system requires the design of the double precision floating point adder/-subtracter, the design of the LPROM, the creation of the fixed ROM, and the integration and testing of the complete system. The complete LPASP is the largest and most powerful chip ever designed at the Air Force Institute of Technology.

II. Background

2.1 Introduction

The idea for the LPASP was created through the work of Capt Richard Linderman, Capt David Gallagher, and Capt John Comtois. Capt Linderman created the initial architecture for a single precision integer ASP and integrated the idea of combining an ASP with a laser programmable ROM. Capt Gallagher created a library of sub-circuits for a bit-slice application specific processor as part of his thesis effort at AFIT(9). The library of sub-circuits was modified by Capt Comtois for use in the LPASP. Capt Comtois also took the single precision ASP architecture and built from it the LPASP double precision architecture.

Capt Comtois created an ASP with four major components(4). The largest component was the double precision floating point multiplier. The multiplier was accompanied by a double precision floating point adder/subtracter, a microcode store, and the integer datapaths with control logic. This chapter will take a brief look at the history behind each major component.

2.2 Multiplier

The architecture and the initial layout of the mantissa for the double precision floating point multiplier was developed by Captain Erik Fretheim(8). A subset of the initial cells for the mantissa were tested as part of an EE695 class project. The mantissa cells were tested with ESIM and some errors were found in the logic for the octal Booth encoding. The erroneous cells were corrected by the testing team, and a small mantissa was created with the new cells, but a test chip was not fabricated. The completion of the multiplier is the highest risk item for the completion of the LPASP.

2.3 Adder

The double precision floating point adder design evolved from an architecture that Major Joe DeGroat created and simulated using the VHSIC Hardware Description Language (VHDL). The initial layout for the adder became a class project for Captain Charles Wardin and myself(16). The adder was completed as part of this thesis project and is discussed in detail in Chapter 4.

2.4 Microcode Store

The LPASP contains two ROMs that are used to store the microcode which contain the control signals for the tasks to be executed by the LPASP. The first ROM is a fixed ROM, which is created

by the AFIT XROM compiler. The fixed ROM holds the common LPASP routines to perform dot product, matrix multiplication, etc. Each fabricated LPASP contains the same microcode in the fixed ROM section. The second ROM is the laser programmable ROM (LPROM). An LPROM architecture and design was initiated by Captain John Tillie using a diffusion link technique(19). The diffusion link technique was created by Lincoln Laboratories where it has been employed very successfully(14). Unfortunately, Capt Tillie could not get his LPROM test chip to work. The re-evaluation of laser techniques and the architecture and design of a new LPROM is part of this thesis effort.

2.5 Datapaths with Control Logic

After the architecture for the LPASP was completed, Capt Comtois began designing macrocells for the LPASP. He completely designed the integer datapaths that contains two 32-bit integer arithmetic logic units (ALUs), a barrel shifter, two sets of 25 storage registers, six incrementers, and four address pointer/incrementer registers. The chip is laid out such that there is an upper datapath and a lower datapath. He also created the control logic macrocells to control the instruction addressing, branching, and the stack. The architecture and the cells created by Capt Comtois will be discussed in detail in Chapter 3.

III. Architecture of LPASP

3.1 Introduction

The architecture for the LPASP was created by Capt John Comtois as part of his thesis effort(4). He started with an architectural specification and a proposed microcode instruction set. He also received input from the Summer term EE588 class, which was tasked to write the microcode for a Kalman filter algorithm. The information for this chapter comes exclusively from Chapter 3 of Capt Comtois' thesis.

3.2 Data Representation

The double precision floating point data representation requires 64 bit data words, but the only macrocells that have to use the double precision format are the floating point multiplier and the floating point adder/subtracter. The integer datapath and address control units could have been designed to use 64 bit data words, but a 64 bit ALU would require more time to execute than a 32 bit ALU and the memory address requires only 20 bits. The extra bits would only slow down the LPASP. The solution was to use 32-bit data words for integer operations and 64-bits data words for floating point operations. The data format for double precision is shown in Figure 3.1.

3.3 General Architectural Features

To support the floating point units, two 32-bit datapaths were created that could combine to drive the floating point units or work in parallel on integer data. The I/O data width was set to 64-bits with the external memory split into upper and lower 32-bit halves. Each memory half is independently addressable; such that, the upper datapath can access memory at the same time that the lower datapath is accessing memory at a different address. The upper and lower datapaths are mirror images of each other with the exception of the barrel shifter in the lower datapath and the function ROM in the upper datapath (see Figure 3.2). The function ROM is only used by the floating point units and is therefore required on only one of the datapaths. The barrel shifter was placed only on the lower datapath because of its size. The upper datapath can access the barrel shifter by using the bus ties.

3.4 Bus Architecture

Each of the LPASP's datapaths contain five data busses (see Figure 3.2). The A and B busses are primarily used to provide input data to the processing hardware. The A bus can be

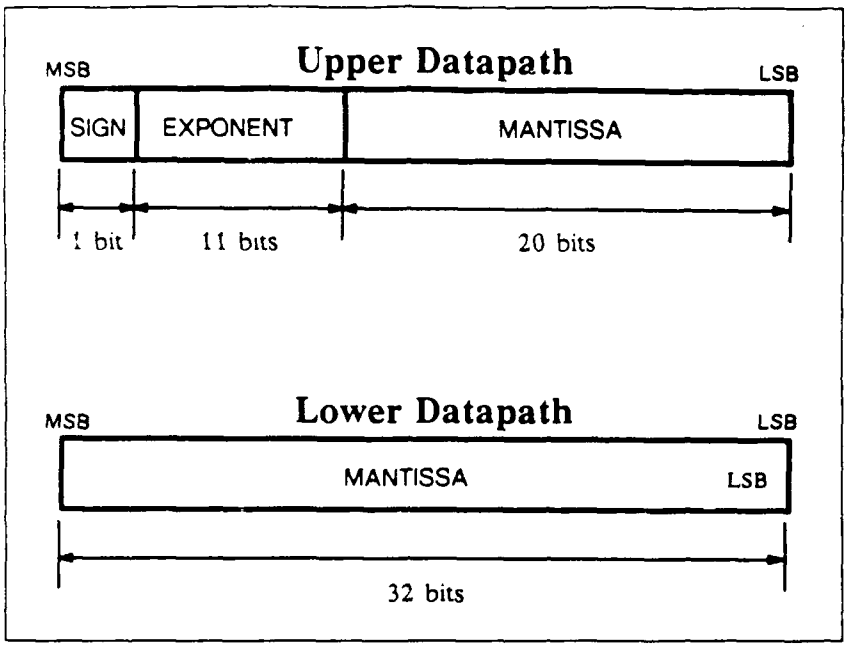

Figure 3.1. Double Precision Representation Inside the LPASP.

driven by any register except the Memory Address Register (MAR). The B bus can be driven by the general purpose registers, the incrementable data registers, or the floating point units. The C bus is primarily used to return the results of the processing units. The C bus can drive any of the registers. The D bus is a bidirectional bus which can be driven by the MBR for a data write to memory or driven by the data pads for a memory read to the MBR, general purpose register 1 (R1), or general purpose register 2 (R2). The E bus is an address bus that can be driven by the least significant twenty bits of either of the pointer registers and can drive the MAR. The E bus allows the A, B, and C busses to be used by the processing units while the pointer registers use the E bus to manipulate the MAR.

The busses of each datapath can operate within there associated datapath or can be independently tied to their counterpart bus. This allows data from the upper datapath to be transferred to the lower datapath or vice versa. It also allows data from one register to drive both datapaths at the same time. The number of operations may be limited to one when the busses are tied together. This ability is especially useful for reading and writing floating point data because the E bus can be tied together to drive both MARs with the same address. The MARs can then direct the memory to store or read the upper and the lower portions of the floating point data from the same memory location.

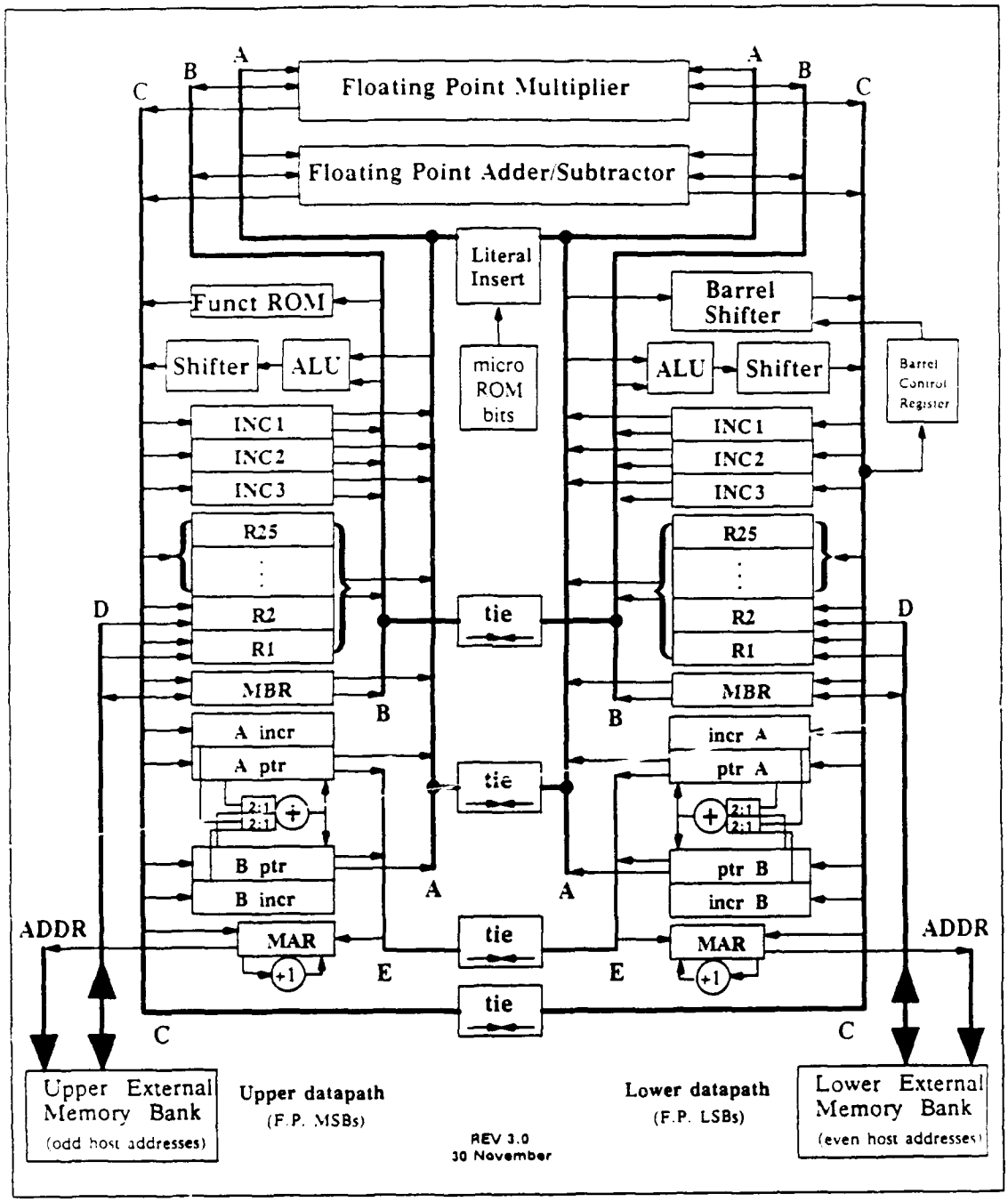

Figure 3.2. LPASP Register-level Description.

3.5 *Register Types*

Five bits were set aside in the microcode for register selection. The five bits of microcode allow for 31 registers and a NOP selection. Of the 31 registers, there are five different kinds of registers in the LPASP.

3.5.1 Incrementable Registers. The first register type is an incrementable register. The incrementable register allows the contents of the register to be unchanged or incremented by one each clock cycle. There are three incrementable registers in each datapath, which were included to allow looping without burdening the ALU. Each register includes a zero flag for testing conditional branches. Between the two datapaths, loops can be nested six deep without impacting the ALU.

3.5.2 Pointer Registers. The second type of register is the pointer register. There are two pointer registers and an adder in each datapath. Each pointer register has an associated register that hold an increment value. During a clock cycle, one of the pointer registers can be incremented by the value stored in its increment value register. The pointer registers were included in the architecture to allow for data retrieval of columns of data in a matrix. The next element in a row is located at the next address, but the next element in a column depends on the number of elements in the row.

3.5.3 Memory Registers. Each datapath contains two registers used to manipulate memory. The first register is the 20-bit Memory Address Register (MAR). The MAR is used to point to the location in memory that is to be read or written. The MAR can be loaded by the E bus, the 20 least significant bits of the C bus, or the value of the MAR incremented by one. The second register is the Memory Buffer Register (MBR) which is used to hold the value that is to be stored in memory. It can also store the value that is read from memory, but R1 and R2 have also been given this capability so that a value from memory can be read into R1 or R2 through the D bus at the same time that a resultant value is loaded in the MBR from the C bus. In the next clock cycle the resultant value can be stored in memory.

3.5.4 General Purpose Registers. The remaining 25 registers were made into general purpose registers: R1 through R25. The general purpose registers are 32-bits wide and can drive either the A bus or the B bus. They are loaded by the C bus, with the exception of R1 and R2 which can be loaded by the C bus or the D bus.

The idea of the LPASP is to provide processing capabilities for a wide variety of algorithms. The best way to provide a variety of processing capabilities is to include a wide variety of processing components. The LPASP contains seven different processing components. Integer processing is handled by the integer ALU, the bidirectional linear shifter, the barrel shifter, and the literal inserter. Floating point processing is achieved by the floating point multiplier, the floating point adder/subtracter, and the function ROM

3.6.1 Integer Arithmetic and Logic Unit. Each of the datapaths in the LPASP contains an 32-bit integer arithmetic and logic unit (ALU). The ALU can perform almost all of the common arithmetic and logic functions, expect for multiplication, which is handled as an option of the floating point multiplier. The functions performed by the ALU are listed in Table 3.1. The "add with carry" (ADC) and "subtract with borrow" (SWB) functions were included to allow arithmetic operations on more than 32 bits. The ALU is driven by the A and B busses and drives the bidirectional linear shifter. Each of the ALUs has a set of independent flags that include a carry/borrow flag, an overflow flag, a zero flag, and a sign flag (see Figure 3.3). The flags can be used for branching in the clock cycle following the cycle where they are set.

Function	**Value Passed to Shifter**	**Flags affected** CARRY, OVERFLOW, SIGN, ZERO
MOVN	A	none
OR	A OR B	zero
AND	A AND B	zero
XOR	A XOR B	zero
MOV	A	zero
NAND	A NAND B	zero
NOR	A NOR B	zero
NOT	NOT A	zero
INC	A + 0 + 1	All Four
SET	A + B + 1	All Four, Sets CARRY
ADC	A + B + previous carry	All Four
ADD	A + B + 0	All Four
NEGA	$\overline{A}$ + 0 + 1	All Four
SUB	A + $\overline{B}$ + 1	All Four
SWB	A + $\overline{B}$ + previous borrow	All Four
DEC	A + 1 + 0	All Four

Table 3.1. ALU Operations.

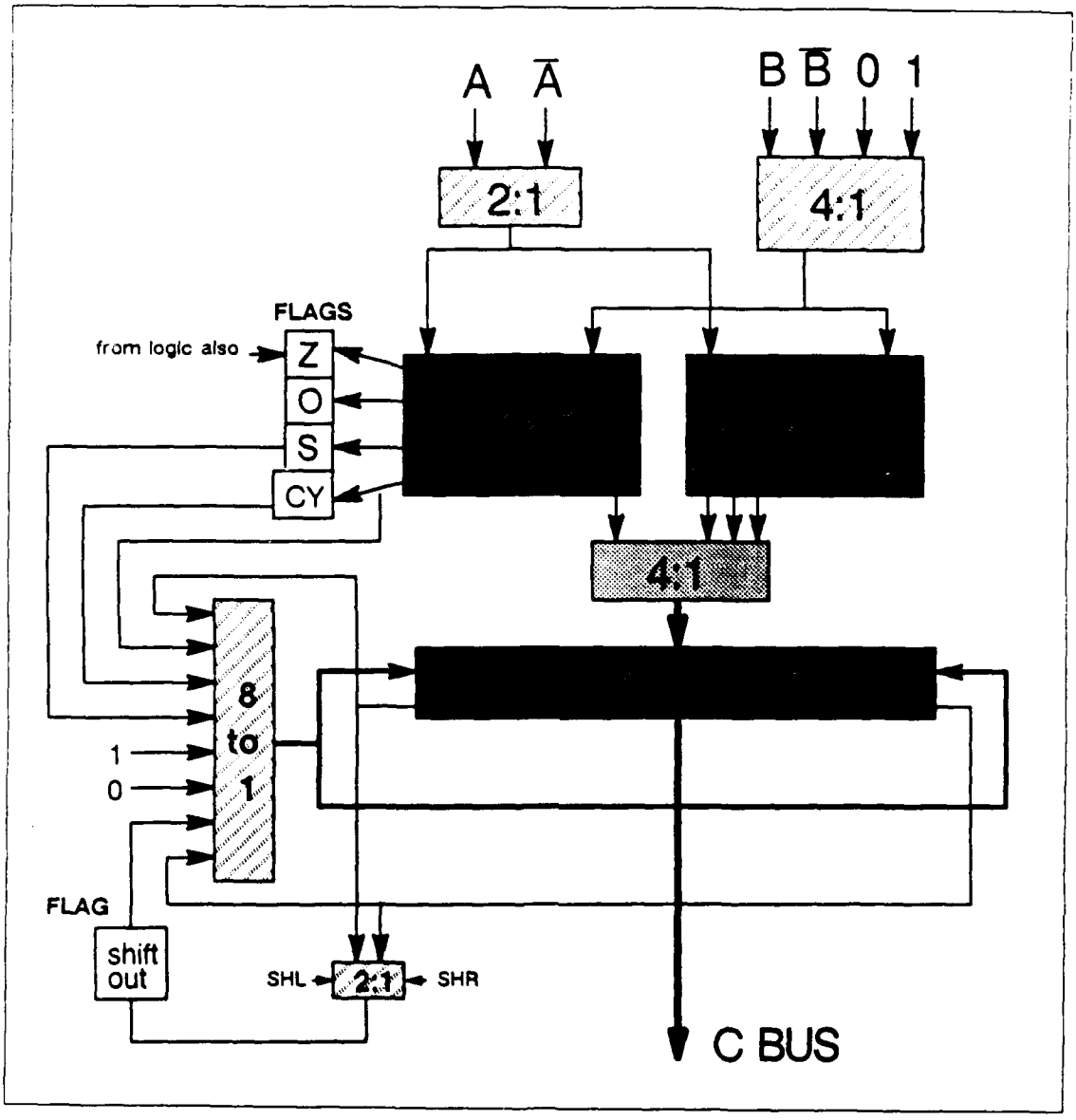

Figure 3.3. ALU and Linear Shifter.

3.6.2 Bidirectional Linear Shifter. The bidirectional linear shifter can perform a one bit shift in either direction or pass the data without modifying it. Whenever a shift is performed, the bit shifted out is stored for the next ALU operation. The shifter can perform arithmetic shifts, logical shifts, circular shifts, and shifts using the stored bit from the previous ALU operation (see Table 3.2). The shifter is driven by the output of the ALU and drives the C bus (see Figure 3.3. The shifter can also ground the C bus, which is useful for clearing registers without using the A or B busses.

Function	Type of Shift	Bit Shifted In
NOP	Shifter does not drive C bus	none
GNDC	Shifter grounds C bus	none
PASS	No shift, ALU output goes on C bus	none
SLOT	Chained Left shift	previous shift-out bit into LSB
SLMS	Circular Left shift	MSB circulated into LSB
SLCY	Shift Left with Carry	Carry of present ALU operation
SL0	Shift Left with Zero	0 into LSB
SL1	Shift Left with One	1 into LSB
SRLS	Circular Right shift	LSB circulated into MSB
SRCF	Shift Right with previous Carry	Carry flag into MSB
SRS	Shift Right with previous Sign	Sign flag into MSB
SROT	Chained Right Shift	previous shift-out bit into MSB
SRSE	Arithmetic Right shift	MSB extended
SRCY	Shift Right with Carry	Carry of present ALU operation
SR0	Shift Right with Zero	0 into MSB
SR1	Shift Right with One	1 into MSB

Table 3.2 Shifter Functions.

3.6.3 Barrel Shifter. The barrel shifter is a left circular shifter capable of shifting from 1 to 31 bits in one clock cycle. Because of its size, the barrel shifter is only located on the lower datapath. It receives it input from the A bus and outputs to the C bus. The upper datapath can drive the barrel shifter if the busses are tied, but the barrel shifter cannot pass data unshifted. The barrel shifter is controlled by the shift length register, which can be loaded from the microword control bits or the lower five bits of the C bus. The shifter can be tricked into acting like a linear shifter by using the literal inserter to zero out the most significant bits that are shifted around. This requires that the number of shifted bits be known in advance.

3.6.4 Literal Inserter. The literal inserter takes 16 bits from the microcode word and places them on the LSBs or the MSBs of either A bus. The literal inserter allows constants in an algorithm without using memory or general purpose registers. Thirty-two bits of data can be inserted by

placing the most significant 16 bits in the LSB of the A bus and then sending them through the barrel shifter to shift them to the MSBs. The least significant 16 bits can be placed with the literal inserter during the next clock cycle.

3.6.5 Function ROM. The function ROM is used to store seeds for iterative subroutines. The seeds help routines such as square root or inverse to converge quickly. The ROM can store tables of seeds for eight different routines. The table selection is accomplished with control bits, while the table index uses the data on the B bus. The function ROM is located on the upper datapath so that the output drives the exponent and the MSBs of the mantissa of the floating point number. The function ROM is divided into a pre-programmed section and a laser programmable section so that user specific seeds can be stored in the ROM. The output of the ROM goes to the C bus.

3.6.6 Floating Point Multiplier. The floating point multiplier is used to multiply 64-bit floating point numbers or 32-bit integer numbers. It is the largest macrocell in the LPASP and requires two clock cycles to settle. The multiplier receives its inputs from the A and B busses and can output to either the C bus or the B bus. For floating point operations, the most significant 32 bits come from the upper datapath, and the least significant 32 bits come from the lower datapath. The multiplier supports the IEEE double precision floating point format, including the flagging of the conditions of underflow, zero result, denormalized result, overflow, and a not-a-number result. An additional flag, called the TRPS flag, is the OR of those flags that indicate an invalid result. The B bus output was created to allow the output from the multiplier to immediately drive the input to the floating point adder. The B bus connections allows the user to alternate driving the multiplier and the adder so that a floating point operation can be initiated every clock cycle.

3.6.7 Floating Point Adder/Subtracter. The floating point adder/subtracter is used to add or subtract two 64-bit floating point numbers. It receives inputs from the A and the B busses and outputs to either the C bus or the B bus. It supports the IEEE double precision floating point format, and can drive the floating point multiplier directly through the B bus. The interface to the adder/subtracter is the same as the floating point multipliers. The adder will be discussed in greater detail in Chapter 4.

3.7 Microsequencer

The microsequencer controls the addressing of the ROMs that hold the microcode instructions. At the center of the microsequencer is the Control Address Register (CAR) that contains the address for the microcode ROMs (see Figure 3.4). The CAR is a 10-bit register that is loaded every clock

cycle from a 4-to-1 multiplexer. The multiplexer chooses between the incremented value of the CAR's present contents, an address from the microcode word presently on the control bus, the address on the top of the stack, or an address from the mapping ROM. The CAR is zeroed by the GO signal to initialize the CAR to the beginning of the microcode routine. With ten bits in the CAR, the ROMs can contain 1024 words, though they only use 784.

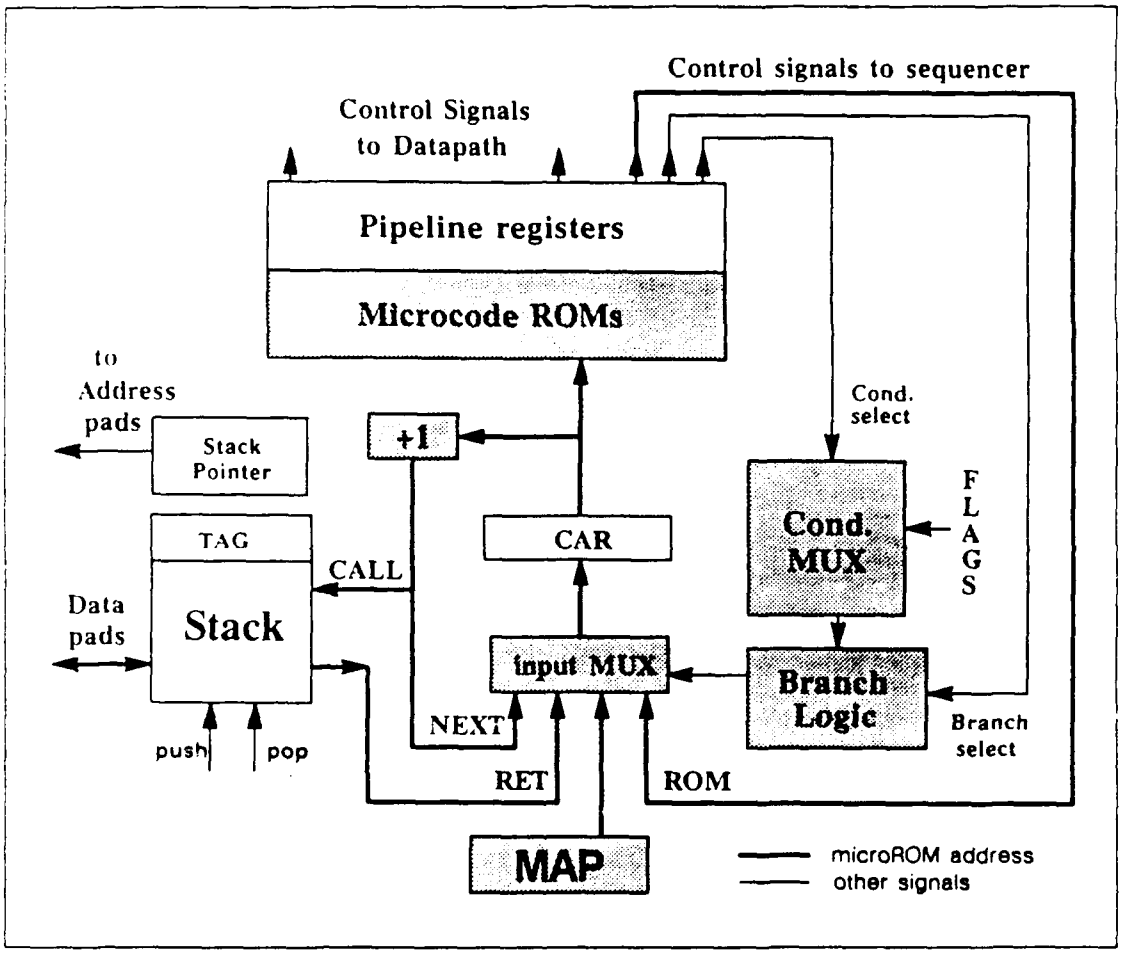

Figure 3.4. Microsequencer Block Diagram.

The four selections were chosen to support sequential flow, branching and subroutine calls, subroutine returns, and assembly language support, respectively. The default selection is the incremented value for sequential flow. The incremented value is also selected when the branch logic is false. The branching logic is handled by a 48-to-1 multiplexer, but flags to one section of the mux can be inverted, which gives a total of 64 branch conditions. The conditions include "uncondi-

tionally true" and "unconditionally false" (see appendix A.1 of Comtois). A subroutine call pushes the incremented value onto the stack and then performs the branch. A subroutine return pops the return address from the stack. The final selection uses the bits from the lower R1 register as an address into the mapping ROM to select and address used by the assembler.

After the address is placed in the CAR, the microcode instruction selected by the address in placed into the instruction pipeline. The pipeline provides an orderly presentation of the control bits to the datapath, but it also creates a one clock cycle delay between the time the address is chosen and the time its microcode control word is executed. This inherent delay means that one instruction is always executed following a branch instruction before the branch can occur.

3.7.1 Microaddress Stack Architecture. The stack used to store the return addresses for subroutine calls has a limited size. If the number of nested subroutine calls exceeds the size of the stack, then the lowest address on the stack is sent to the lower data pads and an address from the external stack pointer is sent to the lower address pads. The stack architecture allows for the stack to overflow into external memory (see Figure 3.5). When the stack is popped, the address from the external stack is placed back onto the internal stack. The external stack pushes and pops do not effect the MAR or the MBR of the lower datapath or any of the lower datapath operations except that memory reads and writes on the lower datapath can not be performed at the same time as a subroutine call or subroutine return. Two counters are used to keep track of the address and status of the external stack.

3.7.2 Microcode Storage. The microcode words are stored in two different types of ROMs to support library routines and user specified routines. The fixed ROM is a silicon compiled optimized ROM that was created at AFIT by Capt Rossbach [Ros85]. The fixed ROM stores common library routines to support dot product, matrix multiplication, etc. It is presently designed to hold 640 words of microcode. The second ROM is a laser programmable ROM (LPROM). The LPROM is where the user stores personalized routines to perform a specific task. The LPROM routines can call the library routines that are stored in the fixed ROM or call routines that are placed in the LPROM. The LPROM was initially design by Capt Tillie as part of his thesis effort [Til88] and was sized at 144 words. A new LPROM was designed as part of this thesis effort that is twice as dense as Capt Tillie's design, so the size will double to 288 words. The microcode word length is 128 bits, which would result in very slow access times for the ROMs, so the ROMs are divided into upper and lower 64-bit halves. The upper half contains control bits for the upper datapath, while the lower half contains control bits for the lower datapath.

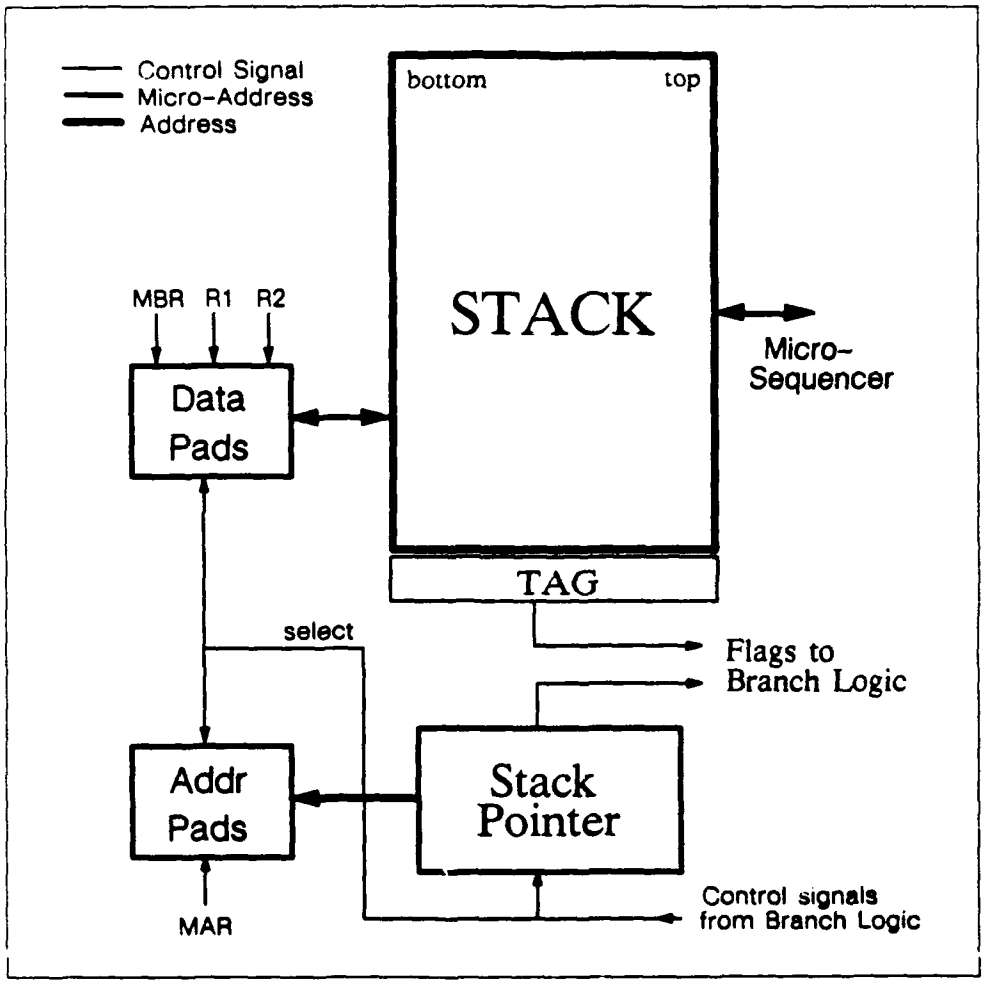

Figure 3.5. Stack Architecture.

3.7.3 Microinstruction Pipeline. The pipeline register was included in the architecture of the LPASP for two reasons. First, it allows more time for decoding the control bits and letting the precharged output of the ROMs to settle. Second, it was designed to allow external testing of the contents of the pipeline. The register is designed to perform as a normal parallel-in/parallel-out register under normal operations, but it can also act as a parallel-in/serial-out register that outputs to a pad and inputs from a pad serially. The register is on a different clock from the rest of the chip so that testing can be performed without affecting the rest of the chip. Also the input and output pads are different so that the serial output can be wired to the serial input, and the contents of the register can be tested without being destroyed.

3.8 Assembly Language Support

After some code was written for the LPASP, it was discovered that 144 words may not be enough to store all of the users routines. The solution was to support an assembly language. The assembly language works by using both of the R1 registers as an instruction register (IR) and the A pointer register as the program counter (PC), with its increment register loaded with a '1'.

The instruction in the IR specifies the source and destination registers, the bus tie configuration (except for the E bus), and the operation code (opcode). "There are six instruction formats. Each format has fields for the replacement control bits needed by the hardware used to execute that instruction"(4).

The PC allows three different addressing modes. The default mode is the incremented value of the PC. The 'indirect' mode uses the an address stored in one of the pointer registers as an address pointer into memory for the address of the final address to be used. The 'immediate' mode uses the 20 LSBs of the upper R1 as an address. One of these addresses is placed into the MAR to obtain the next instruction.

The addresses for the assembler operations are stored in the MAP ROM. The MAP ROM is an LPROM that is addressed by the five opcode bits in the lower R1. It can contain 32 operation addresses, but eleven operations have been predefined. The first operation is the TRAP routine. The ten other predefined operations are defined in Table 3.3. The remaining 21 operations can be programmable by the user.

Instruction	LPASP Operations
Load	load one or two registers from external memory
Store	store one or two register's contents to external memory
Branch	branch to an external memory location on any branch condition, using direct or indirect addressing
Call	call an external subroutine, using direct or indirect addressing
Return	return from an external subroutine
ALU	perform any ALU operation with one or both ALUs with any source or destination registers
Bshift	perform a barrel shift with any source or destination registers
Ptr/Inc	increment a pointer or incrementable register, or load a pointer's increment register
FP+	perform a floating point addition, with any source or destination registers
FP*	perform a floating point multiplication, with any source or destination registers

Table 3.3. Assembly Language Macro Instructions.

3.9 *Remaining Components*

The main emphasis of this thesis is to complete some of the remaining components of the LPASP. There are four major components that have either not been created or their design was incomplete. Of the four remaining components, this thesis has concentrated on the double precision floating point adder and the laser programmable ROM. Test chips for these two components have been designed, fabricated, and tested. The details of the adder and the LPROM is presented in Chapter 4 and Chapter 5, respectively.

The other two remaining components, the double precision floating point multiplier and the fixed ROM, were left in an incomplete state. The multiplier was supposed to be completed by another student in time for the macrocells of the LPASP to be integrated. Unfortunately, the multiplier was not completed in time for this thesis effort. The multiplier, along with the integration of the LPASP, will be the endeavor of a future master's student.

The fixed ROM is the easiest macrocell to create because it is produced by a silicon compiler program. The program receives the data that is permanently stored in the fixed ROM and generates the desired MAGIC files ready to be integrated. The fixed ROM is the last element needed prior to integration because the fixed ROM requires all of the chip resident microcode as input data. The resident microcode can, and probably will, change between now and the time that the chip is fabricated. The creation of the fixed ROM, at this time, would be premature.

IV. Floating Point Adder

4.1 Introduction

The double precision floating point adder is one of the macrocells to be incorporated into the LPASP. The adder receives two 64-bit floating point numbers from the *A* and *B* busses of the LPASP and returns a 64-bit floating point number to the C bus. The adder is designed to process the 64-bit numbers in two clock cycles. It has been initially fabricated as a single chip using 2 micron technology, but will ultimately be fabricated using 1.2 micron technology. This chapter will discuss the details of the floating point adder by presenting a short background of the adder, a detailed discussion of the optimized carry multiplexed (OCM) adder, and a presentation of the components of the floating point adder.

4.2 Background

The double precision floating point adder design evolved from an architecture that Major Joe DeGroat created and simulated using the VHSIC Hardware Description Language (VHDL). The initial layout for the adder became a class project for Captain Charles Wardin and myself(16). The final architecture is a slight modification of DeGroat's original design that includes logic to detect an overshift violation (see Figure 4.1).

4.3 Optimized Carry-Multiplexed Adder

The design of the floating point adder was started by concentrating on the design of the carry select adder because the carry select adder is the largest consumer of the propagation delay through the adder. The result of the analysis of the carry select adder was the invention of the optimized carry-multiplexed adder. The optimized carry-multiplexed (OCM) adder is a new architectural design for large adders. The adder was developed at the Air Force Institute of Technology, and a patent application has been filed. This section starts by presenting the background of the OCM adder, a description of the adder, and a discussion of its performance. The remainder of this section will address the OCM adder cells that were constructed and the OCM test chip that was designed, fabricated, and tested.

4.3.1 Background. The optimized carry multiplexed adder is the product of a three step evolution from the carry-select adder. The first step hierarchically expanded the idea of the carry-select adder into a binary tree structure. The second step broke out the redundant XOR-XNOR and carry chain circuitry that was in each of the full adder cells. The third step broke out the

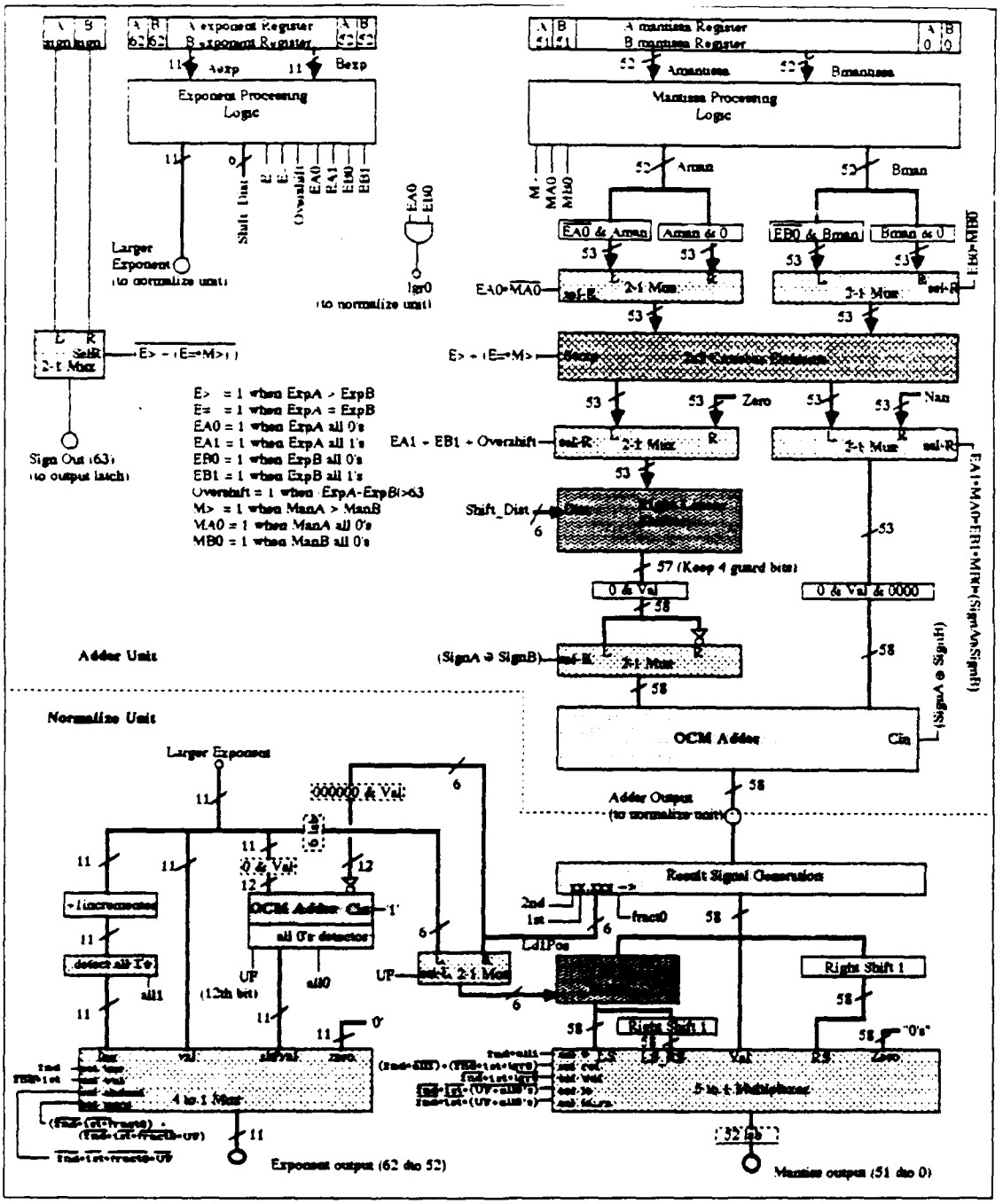

Figure 4.1. Floating Point Adder Architecture

redundant SUM circuitry that was in each of the carry chain cells and fully utilized the parallelism of the binary tree structure.

In July 1988, CPT Erik Fretheim, USA, proposed a carry-select-type adder where the each ripple adder of the carry-select is expanded into a carry-select adder. The idea created a binary tree structure where each level of the tree represents ripple adder chains in parallel. The first level is a ripple adder (see Figure 4.2). The second level is a carry-select adder that consists of two parallel ripple adders, one with an assumed carry input of zero and the other with an assumed carry input of one, and a bank of multiplexers to select the correct sum output (see Figure 4.3). The third level consists of four parallel ripple adders paired into two units, one unit with an assumed carry input of zero and the other unit with an assumed carry input of one (see Figure 4.4). Within each unit is a bank of multiplexers to select the correct sum output for that unit. There is also a bank of multiplexers between the two units to select the correct sum output for that level. The carry out from the level one ripple adder is used as the control for the level two multiplexer bank. The carry out from the upper level two ripple adder is used as the control for the multiplexer bank within the upper level three unit. The carry out from the lower level two ripple adder is used as the control for the multiplexer bank within the lower level three unit. The carry out from the level two multiplexer bank is used as the control for the multiplexer bank between the level three units. The structure is easier to understand if it is realized that the top two levels combine to form a traditional carry-select adder, while the third level is just two carry-select adders in parallel without the initial ripple adders. CPT Fretheim used the idea to create a 56-bit binary tree adder using 4 input ripple adder banks for each of the levels (see Figure 4.5). Each of the thick lines in Figure 4.5 represents a ripple adder bank that consists of four adder cells. The bubbles represent the multiplexers for the carry out. The advantage of the binary tree adder is that the level three structures process twelve pair of inputs in the same amount of time that a carry-select adder would process five pair of inputs. The entire 56 bits can be processed in ten gate delays. The disadvantage of the binary tree adder is that it requires twice the amount of logic as a carry-select adder.

The second step of the evolution to the optimized carry multiplexed adder came from Maj Joe DeGroat, USAF, in November 1988(6). He recognized that each of the adder cells that were being processed in parallel in the binary tree adder contained XOR-XNOR logic with the same inputs. He also realized that the carry chains in the upper level three unit contained the same logic and inputs as the carry chains in the lower level three unit. The only difference between the two units is the input control to the multiplexer banks between the two carry chains. He broke out the XOR-XNOR logic and eliminated the redundant carry chains. The new structure requires only one XOR-XNOR unit and two carry chain units per bit slice whereas the old structure required four XOR-XNOR

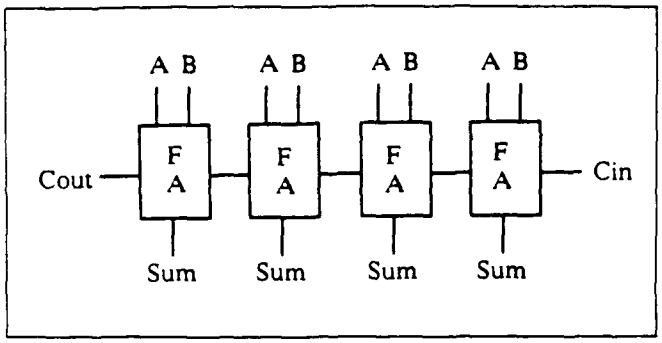

Figure 4.2. First Level

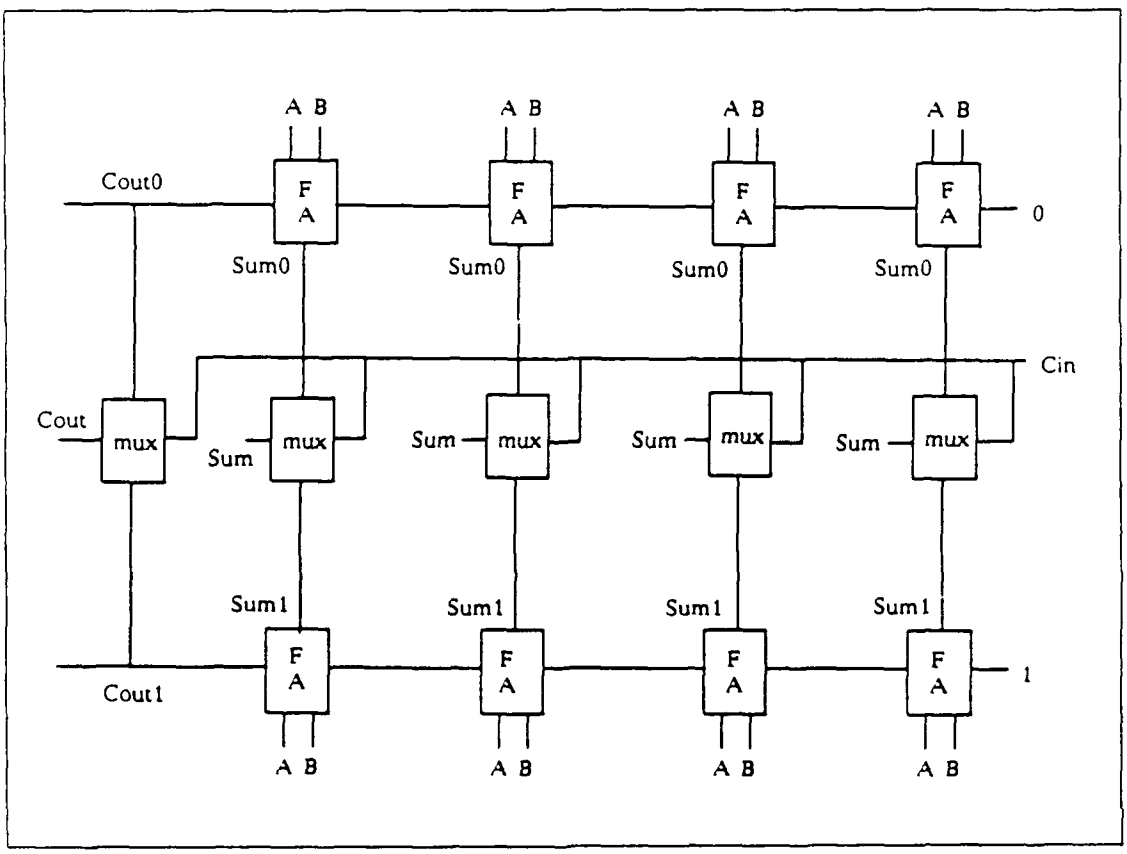

Figure 4.3. Second Level

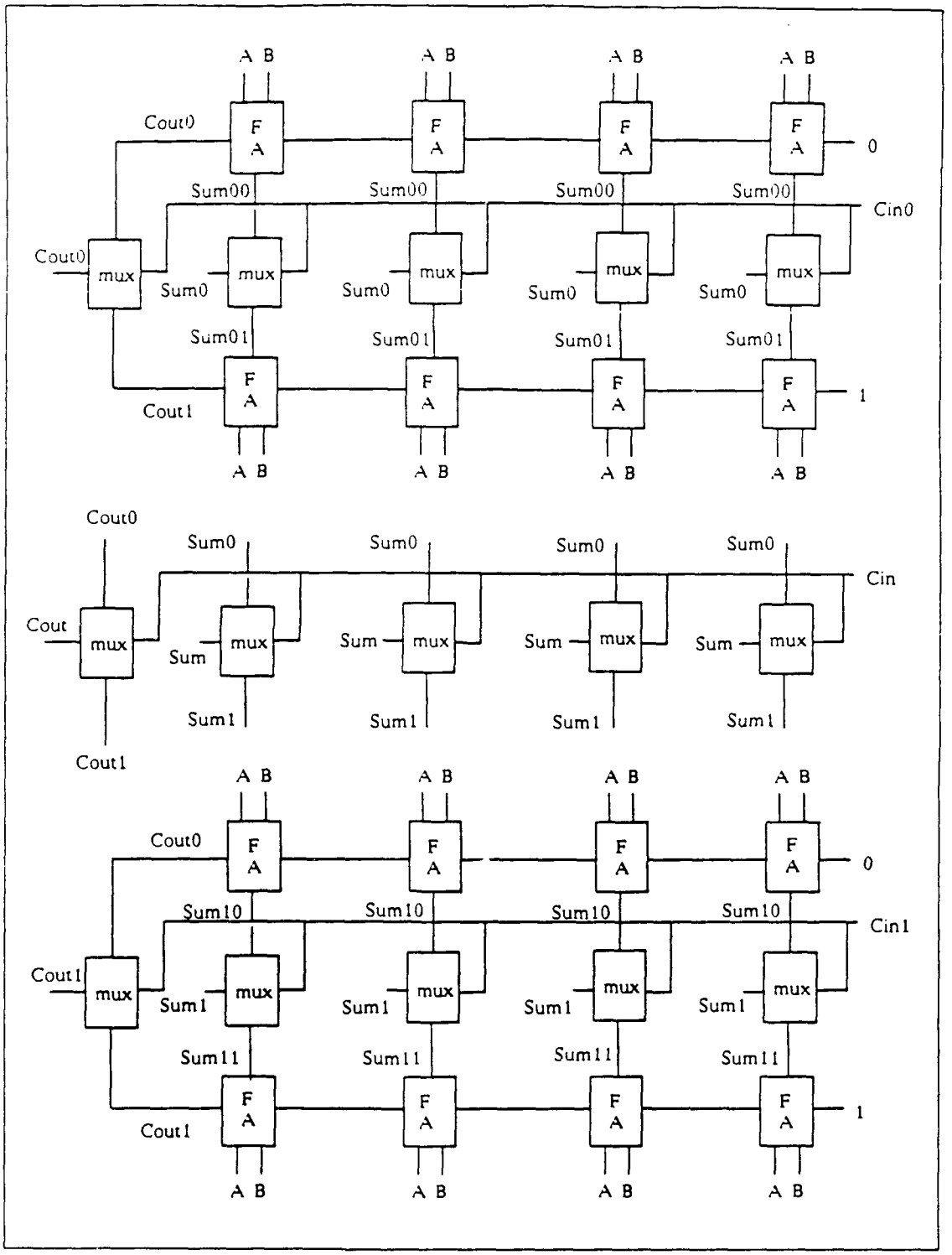

Figure 4.4. Third Level

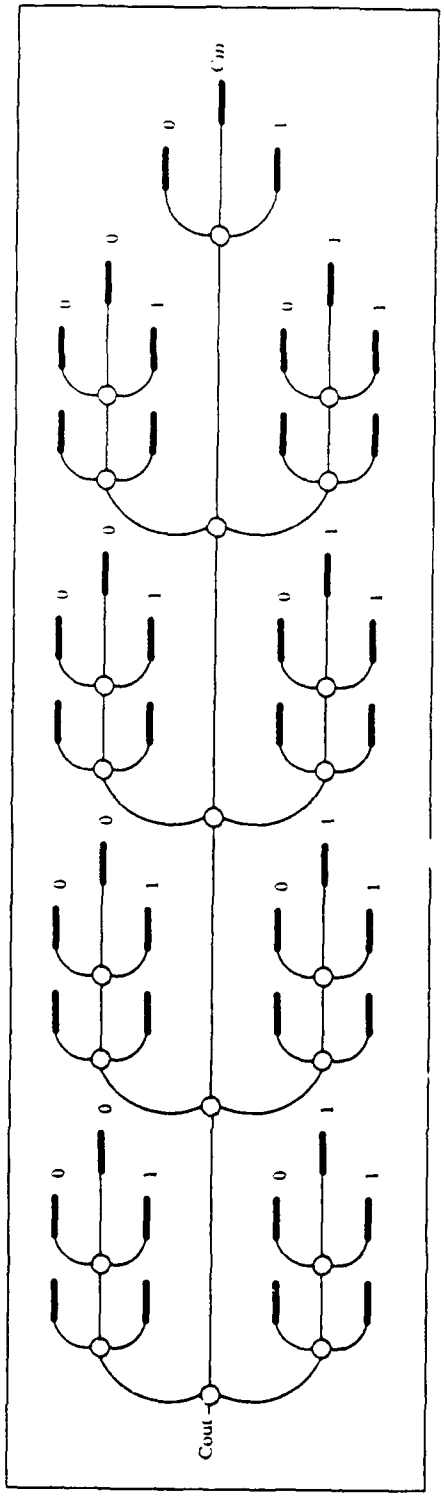

Figure 4.5. 56-bit Binary Tree Adder

units and four carry chain units per bit slice for level three adder cells (see Figure 4.6). The new structure also saved one XOR-XNOR unit per bit slice from level two adder cells. The number of multiplexers per bit slice does not change. With the reduction in logic, the only difference between a level two bit slice and a level three bit slice is the addition of two multiplexers. This modification significantly reduced the amount of required logic without increasing the propagation time.

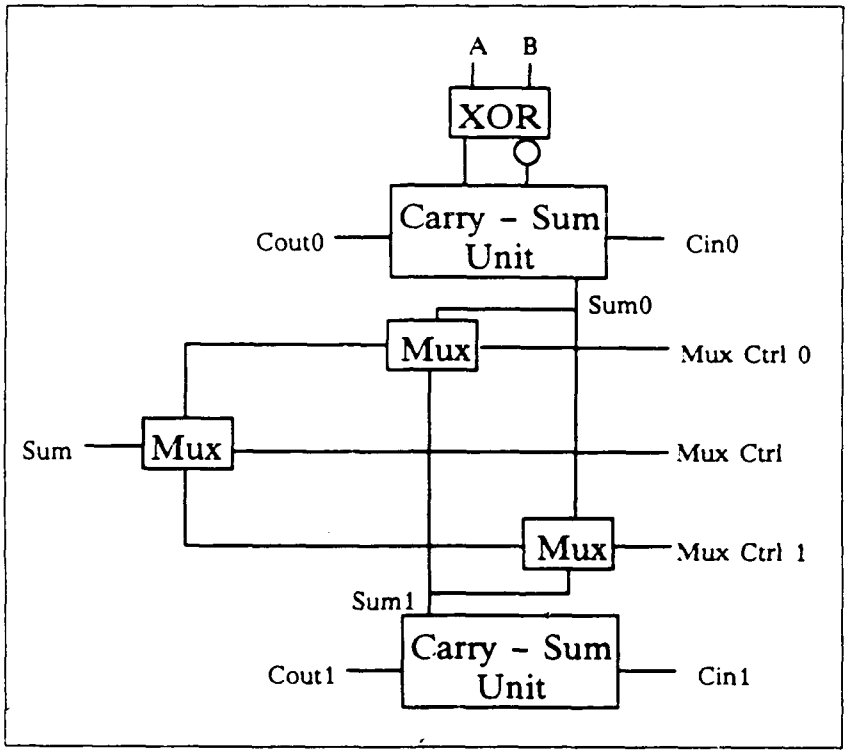

Figure 4.6. DeGroat's Third Level Bit-slice

The final step of the evolution to the optimized carry multiplexed (OCM) adder was an idea of mine in February 1989. I took Maj DeGroat's idea one step further by attempting to break out the sum logic that was contained in each of the carry chains. The problem with breaking out the sum logic is that the two sum logic units did not have the same inputs and the different sum outputs were required as inputs to the multiplexers so that the correct sum could be chosen. The solution to the problem is to use the multiplexers to choose the correct carry output instead of choosing the correct sum output. Then the correct carry output is sent to a single sum unit to determine the sum output (see Figure 4.7). The resulting carry chain unit reduces to just a multiplexer to determine the appropriate carry out (see Figure 4.8). The resulting sum unit is also a multiplexer, but with different inputs (see Figure 4.9). The advantage of this idea is that it requires only one

sum unit instead of two sum units. The disadvantage is that the final output is delayed by one T-gate delay. This delay can be avoided by using DeGroat's version for the final stage of the adder.

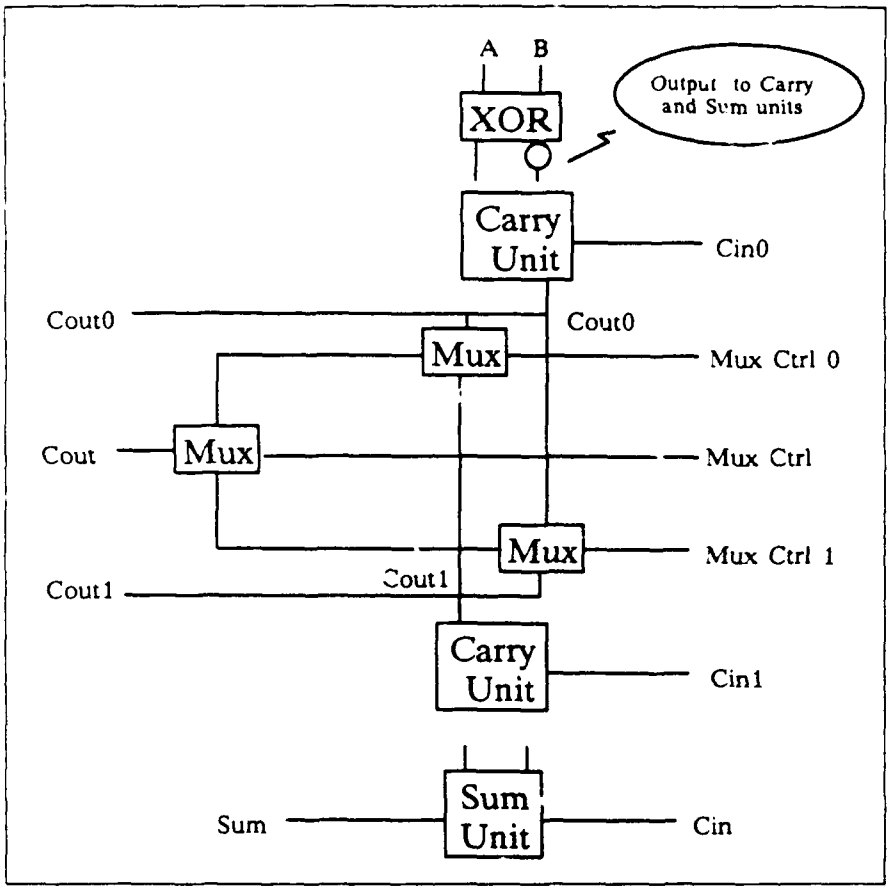

Figure 4.7. Scriber's Third Level Bit-slice

I also investigated the growth properties of the OCM stage. In a traditional carry-select adder, the second level structure is repeated with each successive stage adding another adder cell to the stage to compensate for the multiplexer delay. For the OCM adder, a stage consists of a second level sub-stage followed by multiple third level sub-stages (see Figure 4.10). The entire stage structure is repeated with each succession adding another level three sub-stage to compensate for the multiplexer delay. The OCM adder significantly decreases the carry propagation time for large adders because smaller sub-stages of level three banks can settle in parallel with the largest sub-stage without adding to the carry propagation delay for the stage. For example, if the largest level three sub-stage within a stage is five bits long then a four bit level three sub-stage could feed it, preceded by a three bit sub-stage, a two bit sub-stage, and finally a one bit second level sub-stage

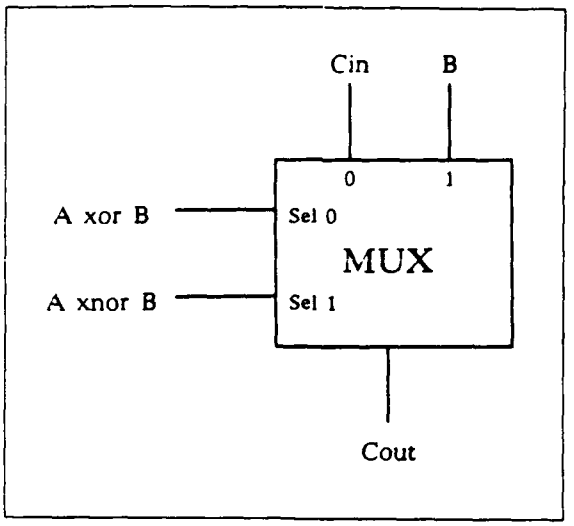

Figure 4.8. Carry Unit

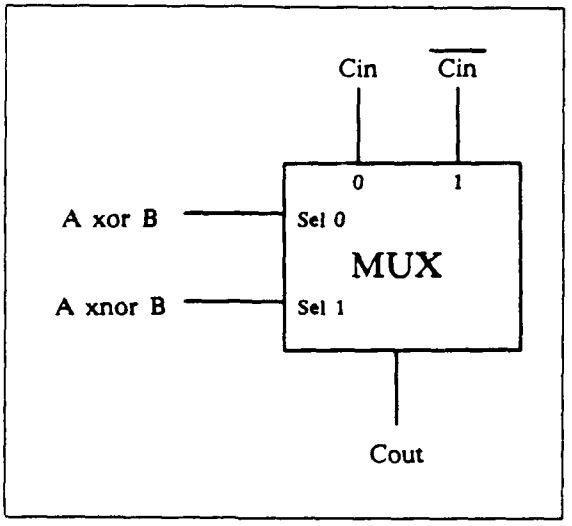

Figure 4.9. Sum Unit

(see Figure 4.11). This fifteen bit stage requires the same amount of propagation time as a six bit stage in a traditional carry-select adder. The stage growth for a OCM adder is quadratic ($\frac{n^2+n}{2}$); whereas, the stage growth for a traditional carry-select adder is linear. Yet, an OCM stage and a carry-select stage settle in the same amount of time.

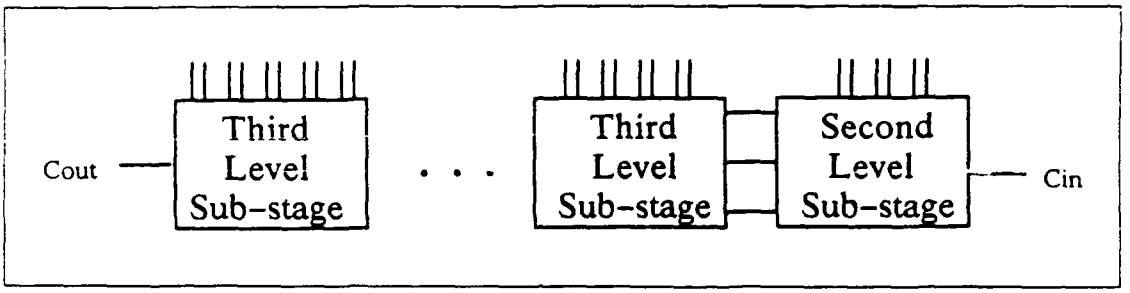

Figure 4.10. OCM Stage

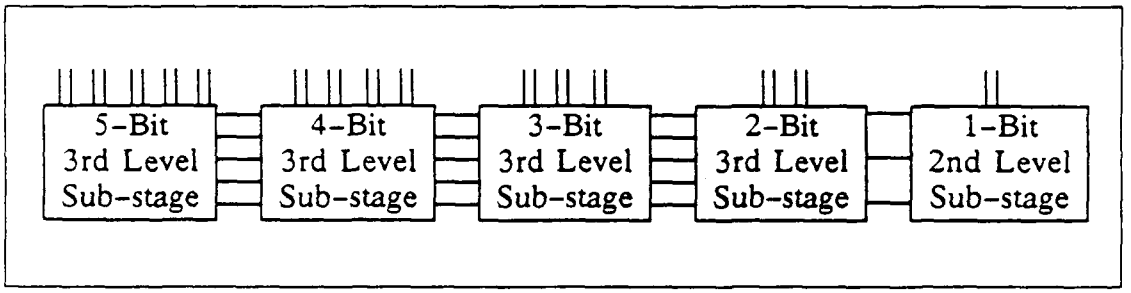

Figure 4.11. Fifteen-bit Stage

4.3.2 Description of OCM Adder. The integration of all of the steps of the evolution results in an adder that consists of multiple stages (see Figure 4.12). The first stage is a standard ripple adder made from standard full adder cells. The carry out of a previous cell feeds the carry in of the following cell. The second stage is a carry multiplexed adder stage whose largest level three sub-stage is equal in length to the first stage. The third stage is identical to the second stage except that it contains an additional level three sub-stage whose length is one greater than the largest level three sub-stage of the second stage. Each successive stage is identical to the previous stage except that it contains an additional level three sub-stage whose length is one greater than the largest level three sub-stage of the previous stage. The final stage is a sum multiplexed adder stage. The difference between a carry multiplexed adder stage and a sum multiplexed adder stage is that each carry unit in a sum multiplexed adder stage contains a sum unit such that the sum output is sent

to the multiplexer(s) instead of the carry output (see Figure 4.13). A carry multiplexed adder stage contains one level two sub-stage and multiple level three sub-stages (see Figure 4.10). The first level three sub-stage has a length that is one greater than the length of the second level sub-stage.

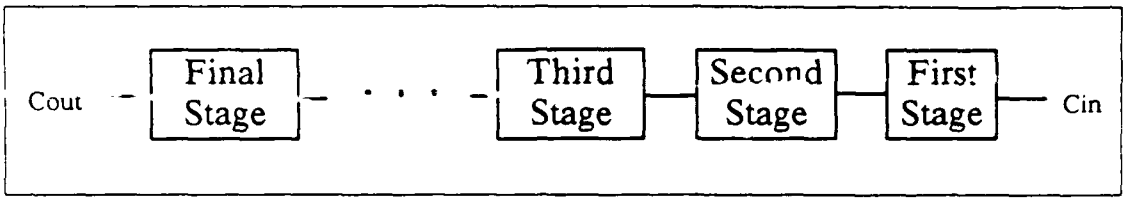

Figure 4.12. OCM Adder

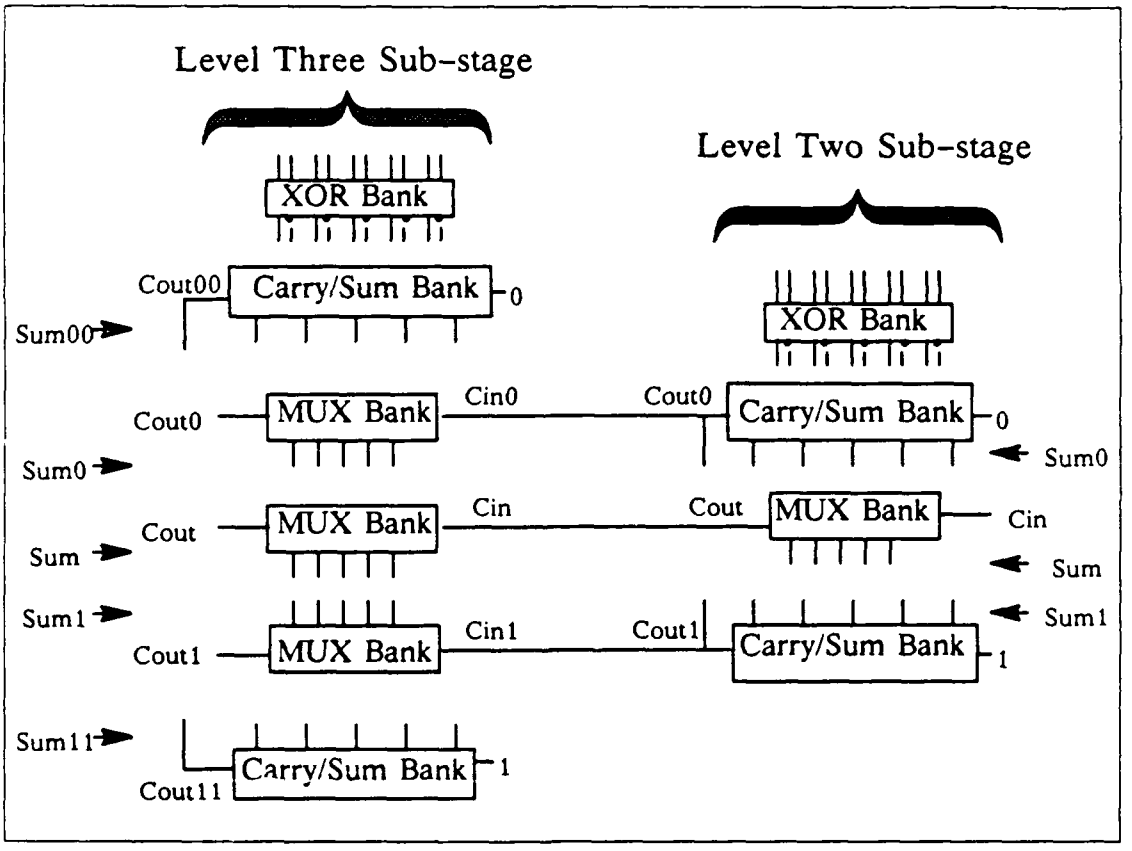

Figure 4.13. Sum Multiplexed Adder Stage

A level two sub-stage is composed of one or more level two adder cells (see Figure 4.14). The first level two adder cell in the sub-stage can be a half adder because it has an assumed carry in of

zero and one respectively to the two carry units. A level two adder cell contains one XOR-XNOR unit, two carry units, one multiplexer, and one sum unit (see Figure 4.15). The XOR-XNOR units receive input from the A and B bits to be added. The carry units receive input from the XOR-XNOR unit, its respective carry in from the previous cell, and an A or B signal. The carry units output C_{out0} and C_{out1} respectively to the multiplexer and to the next adder cell. The multiplexer receives C_{out0} and C_{out1} from the two carry units and is controlled by the carry out from the previous stage. The multiplexer outputs C_{out} to the next adder cell. The sum unit receives inputs from the XOR-XNOR unit and carry in from the previous adder cell.

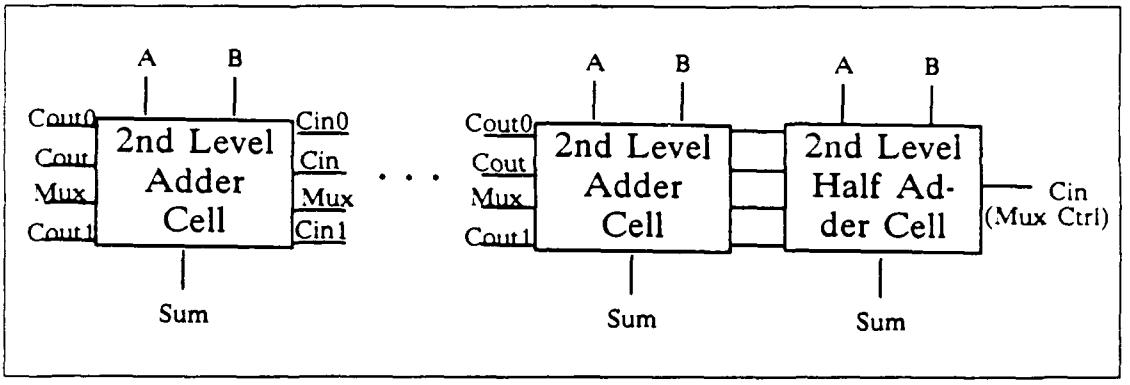

Figure 4.14. Second Level Sub-stage

A level three sub-stage is composed of two or more level three adder cells (see Figure 4.16). The first level three adder cell in the sub-stage can be a half adder because it has an assumed carry in of zero and one respectively to the two carry units. A level three adder cell contains one XOR-XNOR unit, two carry units, three multiplexers, and one sum unit (see Figure 4.17). The XOR-XNOR units receive input from the A and B bits to be added. The carry units receive input from the XOR-XNOR unit, its respective carry in from the previous cell, and an A or B signal. The carry units output C_{out0} and C_{out1} respectively to multiplexer0 and multiplexer1 and to the next adder cell. Multiplexer0 receives C_{out0} and C_{out1} from the two carry units and is controlled by C_{out0} from the previous sub-stage. Multiplexer1 receives C_{out0} and C_{out1} from the two carry units and is controlled by C_{out1} from the previous sub-stage. The third multiplexer receives input from multiplexer0 and multiplexer1 and is controlled by C_{out} from the previous stage. The third multiplexer outputs C_{out} to the next adder cell. The sum unit receives inputs from the XOR-XNOR unit and carry in from the previous adder cell.

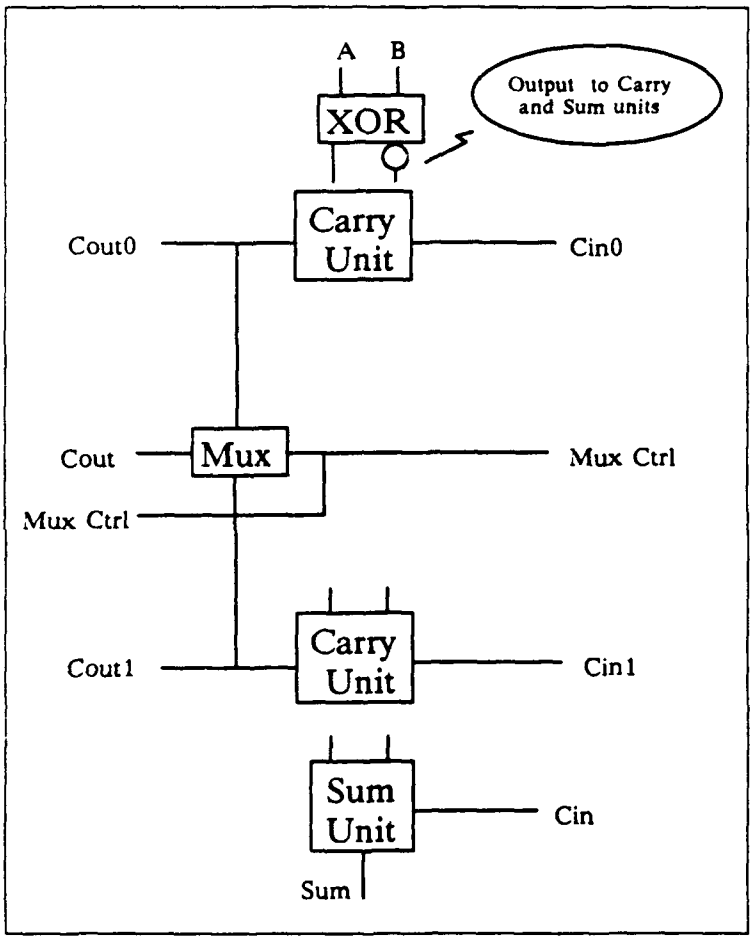

Figure 4.15. Level Two Adder Cell

The interconnections between stages, sub-stages, and cells are very critical. The C_{out} from the final central mux of a stage becomes the mux control signal for all of the next stage. The C_{out0} from the final mux0 of a sub-stage becomes the mux0 control signal for all of the next sub-stage. The C_{out1} from the final mux1 of a sub-stage becomes the mux1 control signal for all of the next sub-stage. C_{out0} and C_{out1} from mux0 and mux1 of a sub-stage are not propagated into the next stage. Within a sub-stage, the C_{out0} from a cell's upper carry unit becomes the C_{in0} for the next cell's upper carry unit. The C_{out1} from a cell's lower carry unit becomes the C_{in1} for the next cell's lower carry unit. The output from a cells two carry units are not propagated to the next sub-stage. The C_{in} for the carry units of the first cell of a sub-stage are 0 for the upper carry unit and 1 for the lower carry unit. In all cases, the C_{out} from a cell becomes the C_{in} for the next cell's sum unit.

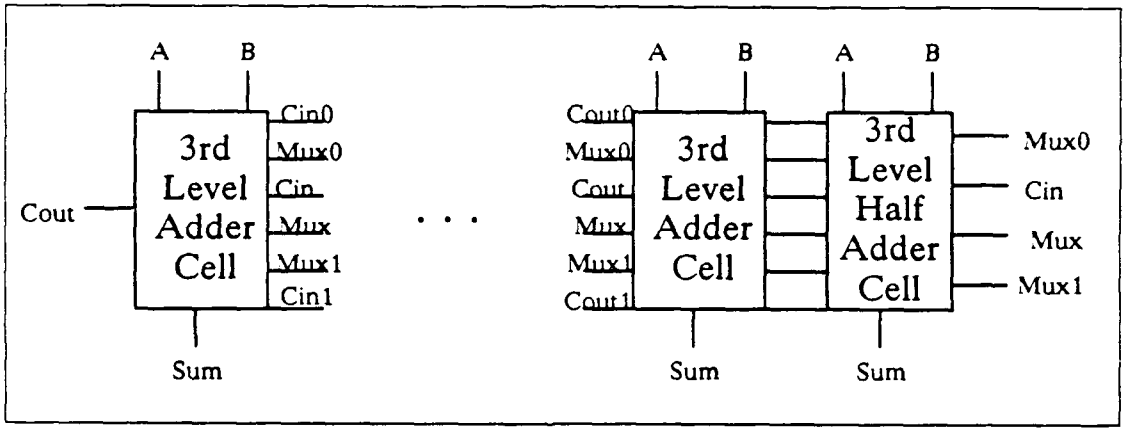

Figure 4.16. Third Level Sub-stage

4.3.3 Performance. The optimized carry multiplexed adder is significantly better than any other adder designed to date. Since the stage growth is quadratic it is a significant improvement over the carry-select adder. Table 4.1 is a comparison of the number of bits that can be executed in a given time for a carry-select adder vs. an OCM adder, assuming each adder starts with a 2-bit ripple adder. The equation for the number of bits processed in a given time for the OCM adder is:

$$\# \text{ bits} = \frac{t^3-t}{6} + 1$$

The equation for the number of bits processed for the carry-select adder is:

$$\# \text{ bits} = \frac{t^2-t}{2} + 1$$

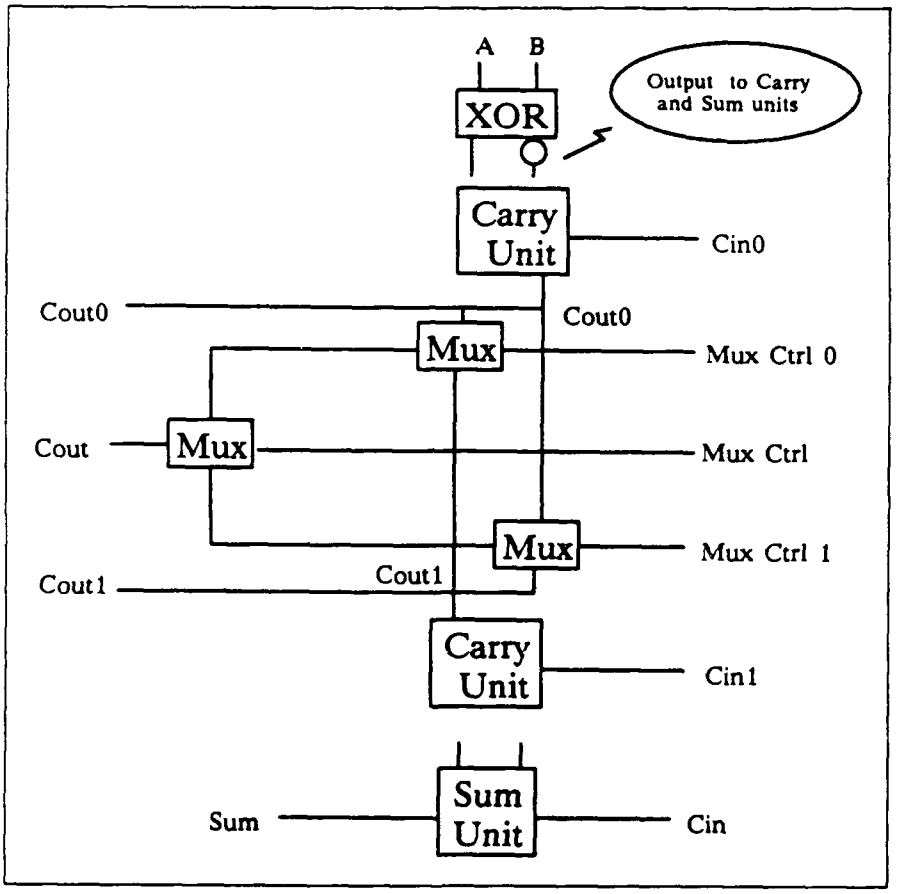

Figure 4.17. Third Level Adder Cell

Gate Delay	Carry-Select bits	OCM bits
4	7	11
5	11	21
6	16	36
7	22	57
8	29	85
9	37	121
10	46	166
11	56	221
12	67	287
13	79	365
14	92	456
15	106	561
16	121	681
17	137	817
18	154	970
19	172	1141
20	191	1331

Table 4.1. Comparison of Carry-Select to OCM

The equation shows that the OCM adder gives cube root performance growth with only linear logic growth. The number of transistors required for a level three adder cell is less than that of a carry-select adder cell. This means that an optimized carry multiplexed adder will require less transistors than a carry-select adder, yet perform significantly faster. The performance improvement depends on the number of bits that will be added together because of the cube root growth characteristic (see Table 4.1, but the future trend seems to be toward words with greater number of bits. With word lengths of 64, 128, or even 256 bits, the optimized carry multiplexed adder will soon be the only way to obtain very high speed adding.

4.3.4 Cell Construction. A set of bit-slice adder cells were constructed using the OCM adder architecture. In the process of producing the cells, an extensive SPICE analysis was performed to optimized the carry propagation time delay. SPICE is a circuit analyzer used to determine timing parameters. Since the final chip will be fabricated using the 1.2 micron technology, the level four Hewlett Packard 1.2 micron SPICE parameters were used for the SPICE simulations for the adder. The results of the SPICE analysis greatly impacted the design of multiplexer cells used by the floating point adder.

Sixteen different adder cells were created for the exponent subtracter while 26 different cells were created for the mantissa adder, but not all of them are used in the floating point adder. All of the cells that might be needed for any design were created, so that the cells could be used

universally in a cell library. The cells were designed to optimize the carry delay and allow stacking of cells such that the carry and mux signals would not require any external routing. The cells use Gallagher's T-gate XOR-XNOR structure, but use N-type pass transistors for the carry logic, the sum logic, and the carry-select muxes. The two types of muxes were SPICEd and it was found that the T-gate mux was faster on average ($t_{DR} = .3254, t_{DF} = .2589$) than the N-pass mux ($t_{DR} = .7820, t_{DF} = .3782$) if the signal was only used to drive gates and was not being inverted or buffered. If the signal was being used to drive another T-gate or was being buffered than it was slower. The reason for the speed difference is the larger drain capacitance for the T-gate and the inverter delay that is required for the N-pass transistor.

The original design for the adder cells immediately inverted the carry-in and inverted the carry-out. This allowed for non-inverted carry signals. The timing results for the optimized cells are presented in Table 4.2.

Changing signal	t_{DF}	t_{DR}
A	1.1568	.7000
B	1.0643	1.0016
C_{in}	.7538	.6835

Table 4.2. Timing Result of Original Cells.

It was discovered that if the carry-in signal was not inverted for the carry propagation then the carry delay was reduced to:

Changing C_{in}	t_{DF}	t_{DR}
	.3416	.6472

The major problem with inverted carry signals is that it more than doubled the number of different cells needed to be designed. The half adder cells were also originally designed with non-inverted carry signals, but were later redesigned with inverted carry signals. The times for half adder cell are shown in Table 4.3. The times for the half subtracter are shown in Table 4.4.

	OLD		NEW	
Changing signal	t_{DF}	t_{DR}	t_{DF}	t_{DR}
A	.6539	.7028	.3410	.3641
B	.6864	.6761	.3714	.3463

Table 4.3. Timing Results for Half Adder.

	OLD		NEW	
Changing signal	t_{DF}	t_{DR}	t_{DF}	t_{DR}
A	.6481	.6931	.4167	.3896
B	.6829	.6258	.3306	.3595

Table 4.4. Timing Results for Half Subtracter.

4.3.5 OCM Adder Test Chip. The OCM adder cells were tested by creating a 57-bit OCM adder. The OCM stages were configured into a (654321, 54321, 4321. 321, 21, 2) architecture. The test adder used the cells created for the OCM adder that is in the mantissa section of the adder unit in the floating point adder. The test adder was fabricated onto a two micron technology tiny chip. A tiny chip is limited to dimensions of 2200 by 2200 microns and is placed in a 40 pin dual-inline package (DIP). See Attachment A for the pin out of the OCM Adder Test Chip.

The limited number of pins and chip area required that the OCM adder be broken into four individually loadable sections. The result is a 57-bit adder chain that snakes across the chip in four sections. The sections are turned onto there neighbor such that the outputs of the two leftmost sections and the two rightmost sections face each other. This layout configuration allowed for one of the four branches of the input bus to be eliminated because the two center sections share a common input bus branch. The layout also conserved space because the output probe pads from the two facing sections on the left and right sides of the chip were staggered into a single alternating array.

Probe pads were used for the outputs because there was not enough room to fit an output data bus nor were there enough pins to output the result.

Each section of the adder has an array of dual master-slave flip-flops (MSFF) to independently latch the A and B data from the input bus. The MSFFs were custom designed to shrink the data pitch down to 48λ and to reduce the usual four control signals down to only two signals(see Figure 4.18). The improvement was accomplished by using the non-inverted control signal to allow data to pass through an N-pass transistor. The signal was then sent through a compensation inverter to restore a high signal. The restored signal is inverted and sent through a P-pass transistor that is controlled by the same non-inverted control signal that controlled the N-pass transistor. The signal from the P-pass transistor is fed back to the compensation inverter, which closes the flip-flop loop. The feedback P-pass transistor was made small because it is not required to change the voltage of the node that gates the compensation inverter. The P-pass transistor is only used to maintain the stored value. When the control signal is high, the input data is allowed through the flip-flop and

the output is the inverse of the input. When the control signal goes low, the input data is cut-off from the flip-flop, but the inverse of the output is fed back into the flip-flop.

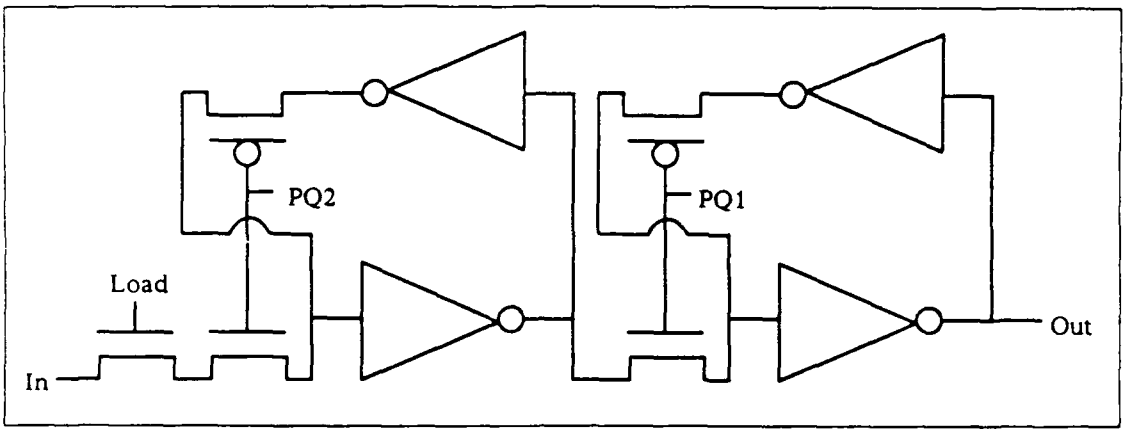

Figure 4.18. Master/Slaves Flip-Flop Architecture

The dual MSFF uses two flip-flops per bit slice and has two bit slices per cell to accommodate for the A and B data signals. The first flip-flop is controlled by a *load* signal that is common to all of the MSFFs in the section. The second flip-flop is controlled by a *go* signal that is common to every MSFF on the chip. To load the 114 bits of input data, the first 15 bits of A and B are placed on the input bus and the load signal for the first section is raised and lowered. Then, the next 14 bits of A and B are place on the input bus and the load signal for the second section is raised and lowered. This process is repeated for the 14 bits of A and B for the third and fourth sections. After all of the slave flip-flops are loaded, the *go* signal is raised and all 114 bits of data are transferred to the OCM adder at the same time.

Four tests were performed on the 57-bit OCM adder. The first test was a power-up test to assure that there were no shorts between power and ground. The chips drew an average current of about 12 milliamps with V_{dd} at five volts. The second test checked the logic of the adder. The third test attempted to measure the time delay through the adder. The fourth test analyzed the noise margin for the MSFFs.

The logic of the adder was tested through two tests. The first test checked the logic of each cell independently. The second test verified that the carry propagated correctly through the adder.

The first logic test was accomplished by wiring the A and B inputs to two switches for the tested cell, wiring the A and B inputs to a single switch for the previous cell, and wiring the A and B inputs to ground for the next cell. The inputs to the previous cell were wired to control the

carry-in to the tested cell. When A and B are zero, the carry must be zero. When A and B are one, the carry must be one. The inputs to the next cell were grounded so that the sum result would equal the carry-out of the tested cell. Probes were connected to the sum output of the tested cell and the sum output of the next cell to evaluate the sum and carry logic of the test cell. All 57 cells passed each of the eight tests that corresponded to the eight possible inputs.

The second logic test was performed by grounding all of the A inputs, except for the least significant bit, A_0, which was wired to a switch. The B inputs were wired to a separate proto-board where each input could be individually raised to 5 volts or grounded. The first part of this test started by setting all of the B inputs high and probing the final carry-out of the adder. The A_0 input was then switched to modify the carry output. The second part of this test verified the carry propagation logic for each of sum outputs from the most significant to the least significant. The inputs were configured the same as in the first part. With A_0 low, the sum output was probed for a high. A_0 was then set high to verify a low sum output. With A_0 switched back to low, the B input for the tested cell was grounded and a low sum was verified. A_0 was then set high to verify a high sum output. The second part of the test confirmed that the carry would only propagate to the first occurrence of a low B input.

The time delay test for the adder was performed using two high impedance, active FET-1 probes attached to an oscilloscope. The input data was set such that all of the Bs were high and all of the As were low, except for A_0, which was connected to a word generator configured to produce a 500 KHz square wave with a 50% duty cycle. The first test was perform by using one probe to monitor the A_0 signal coming from the MSFF and the other probe to monitor one of the 57 sum outputs, S_0 to S_{56}, or the final carry output, C_{out}. The oscilloscope was adjusted to trigger on the rising edge of A_0 so the time delay between the change in A_0 and the desired output could be measured. The results are shown in the Table 4.5. The time delay for this test turned out to be much greater than expected. One reason for the large propagation delays in the larger stages is that the same drivers were used to driver the control signals for the multiplexer in the small stages as in the large stages. The delay can be eliminated by buffering the mux control lines after each sub-stage. This demonstrates, though, that the results within a stage have approximately the same time delay.

The magnitude of the time delay created doubt as to whether the input capacitance of the probe was causing the delay. To test the idea, the next timing test removed the probe from the A_0 signal and placed a standard 500 ohm, 10X oscilloscope probe across the word generator signal outside the chip. The scope was adjusted to trigger on the rising edge of the word generator signal. The time delay for the most significant bits of each of the stages were measured and the results were

OUTPUT	DELAY(ns)	OUTPUT	DELAY(ns)	OUTPUT	DELAY(ns)
S_0	48.5	S_{19}	61.0	S_{38}	78.0
S_1	50.5	S_{20}	63.0	S_{39}	76.0
S_2	51.0	S_{21}	61.0	S_{40}	78.0
S_3	54.0	S_{22}	69.5	S_{41}	76.0
S_4	53.0	S_{23}	70.0	S_{42}	78.0
S_5	52.0	S_{24}	69.0	S_{43}	77.0
S_6	59.0	S_{25}	69.0	S_{44}	75.0
S_7	58.0	S_{26}	69.0	S_{45}	77.0
S_8	58.0	S_{27}	72.0	S_{46}	75.0
S_9	58.0	S_{28}	71.0	S_{47}	76.0
S_{10}	56.0	S_{29}	71.0	S_{48}	76.0
S_{11}	59.0	S_{30}	71.5	S_{49}	75.0
S_{12}	63.0	S_{31}	69.0	S_{50}	74.0
S_{13}	62.0	S_{32}	69.0	S_{51}	78.5
S_{14}	60.0	S_{33}	68.0	S_{52}	77.0
S_{15}	59.0	S_{34}	71.0	S_{53}	74.0
S_{16}	59.0	S_{35}	68.0	S_{54}	77.0
S_{17}	64.5	S_{36}	69.0	S_{55}	76.0
S_{18}	63.5	S_{37}	76.0	S_{56}	79.0
				C_{out}	56.0

Table 4.5. Time Delay from A_0 to Output.

6.5 nanoseconds faster, on average. The time improvement is significant when it is realized that, under the test's configuration, the four inverters from the MSFF were added to the propagation delay, yet the time decreased.

In order to eliminate the capacitive loading of the changing input signal, another test was performed that measured the time delay between the least significant sum, S_0, and other selected sum outputs. The sum outputs are attached to probe pads, but do not drive any other circuit elements. The results of the test are shown in Table 4.6. The least significant sum is produced by an AND gate that should increase the total processing time an additional 1 nsec to about 31 nsec. The propagation time can be improved to about 12 to 15 nsec by properly sizing the multiplexer signal drivers.

The FET-1 probe specifications state that the input capacitance of the probes can range from 0.2 to 5.0 picofarads. A SPICE analysis was performed assuming an 2 nanosecond rise and fall time signal entering the sum circuitry and driving the probe pad and the FET-1 probe. The analysis determined that the sum output rise time could vary from 4 to 61 nanoseconds, and the fall time could vary from 8 to 81 nanoseconds. The measured rise and fall times for S_0 were 20 and 30

SUM OUTPUT	RISE TIME DELAY(ns)	FALL TIME DELAY(ns)
S_1	2.0	-1.5
S_3	3.0	-1.0
S_5	4.0	0.0
S_{11}	5.5	5.5
S_{21}	14.0	9.0
S_{36}	22.0	21.0
S_{56}	30.0	31.0
C_{out}	51.0	9.0

Table 4.6. Time Delay from S_0 to Output.

nanoseconds, respectively. The wide range of the input capacitance explains why the time delays are so great in Table 4.5, and it also explains why some of the values are negative in Table 4.6.

The last test that was performed on the chip was the noise margin test. Since the chip pads buffer all of the input data to the adders, the noise margin test really only tested the noise margin of the chip pads. The test was performed using two power supplies, two volt meters, and a standard micromanipulator probe.

The first power supply was used to set V_{dd} at exactly 5.0 volts. The second power supply was connected to the input pin of the chip as an input to the MSFF. The control lines for the MSFF were set high to allow the input signal to pass through the MSFF.

The ground lines of the power supplies were tied together to form a common ground. The probe was used to measure the MSFF voltage output. The volt meters were configured such that one read the input voltage to the chip pad and the other read the output voltage from the MSFF. The input voltage was slowly dropped from 5.0 volts until the output voltage was no longer 5.0 volts, which occurred at 2.24 volts. Then the input voltage was raised from 1.8 volts until the output voltage was no longer 0.0 volts, which occurred at 2.17 volts. The results conclude that the width of the transitional region is only 0.09 volts. The results of the test are important because the same chip pads were used for the floating point adder and the laser programmable ROM.

4.4 Components of the Floating Point Adder

The floating point adder can be logically divided into an adder unit and a normalize unit (see Figure 4.1). The adder unit shifts mantissas and adds the two numbers together, while the normalize unit normalizes the resulting number.

4.4.1 Adder Unit of the Floating Point Adder. The adder unit receives its data from the A and B busses of the LPASP. It is currently designed to store the values from the A and B busses in a bank of interleaved master-slave flip-flops. A control signal from the microcode determines when data is loaded in the register. The adder unit consists of three sections to process the sign bit, the exponent, and the mantissa of each number (see Figure 4.1). The majority of the processing occurs in the mantissa section where the smaller number is shifted by the difference of the two exponents and added (subtracted) to (from) the larger number.

4.4.1.1 Sign Bit. The sign bits are the easiest to process. Subtraction ($A - B$ only) is performed by complementing the B sign bit. This is accomplished by an exclusive-or of the subtract signal with the B sign bit, which becomes the new value of the B sign bit. The floating point adder macrocell assumes that this logic has occurred before it receives the B sign bit. The A and B sign bits are stored in a master-slave flip-flop and then sent to a 2 to 1 mux which selects the sign bit of the largest number. The mux select logic is $\overline{E_{A>B} + (E_{A=B} * M_{A>B})}$ (see Figure 4.1 for the definitions of the control signals). If the select logic is low then the result sign bit is A's sign bit, and if the select logic is high then the result sign bit is B's sign bit (see Figure 4.19).

4.4.1.2 Exponent. The exponent processing logic receives the eleven bit exponent from the exponent section of the main register. It calculates the difference between the two exponents, determines if A is larger then B ($E_{A>B}$), determines if A and B are equal ($E_{A=B}$), determines if the individual exponents are all zeros or all ones ($E_{A0}, E_{B0}, E_{A1}, E_{B1}$), determines if both A and B are all zeros ($lgr0$), and passes the larger of the two exponents on to the normalize unit.

To calculate the difference between the exponent of A and the exponent of B, the two exponents are subtracted by an OCM adder with the exponent of A bitwise complimented and a carry-in of one. Two subtracters are required since the adder doesn't know which exponent is larger, and it would take too long to calculate the two's compliment of the result if the larger one had been subtracted. The original design was going to use a single $A \oplus B$ unit for both of the adders, but the added complexity wasn't worth designing special cells just for six bits. An additional savings was gained by realizing that if carry-in = 1 then sum = $A \oplus B \oplus 1 = \overline{A \oplus B}$ and carry-out = $A + B$, and it carry-in = 0 then carry-out = $A * B$ (see Figure 4.20). The structure of the first adder, eleven bits wide, is an optimized carry-multiplexed adder with dimensions (43, 4), where the "43" represent the second stage of the OCM adder, which includes a carry select adder of 4 bits driven by a ripple adder of 3 bits and the "4" represents the first stage of the OCM adder, which consists of a ripple adder of 4 bits driving the "43" adder group. The main subtracter is 990λ by 158λ. The second adder, six bits, receives A and B from six additional registers that contain the same

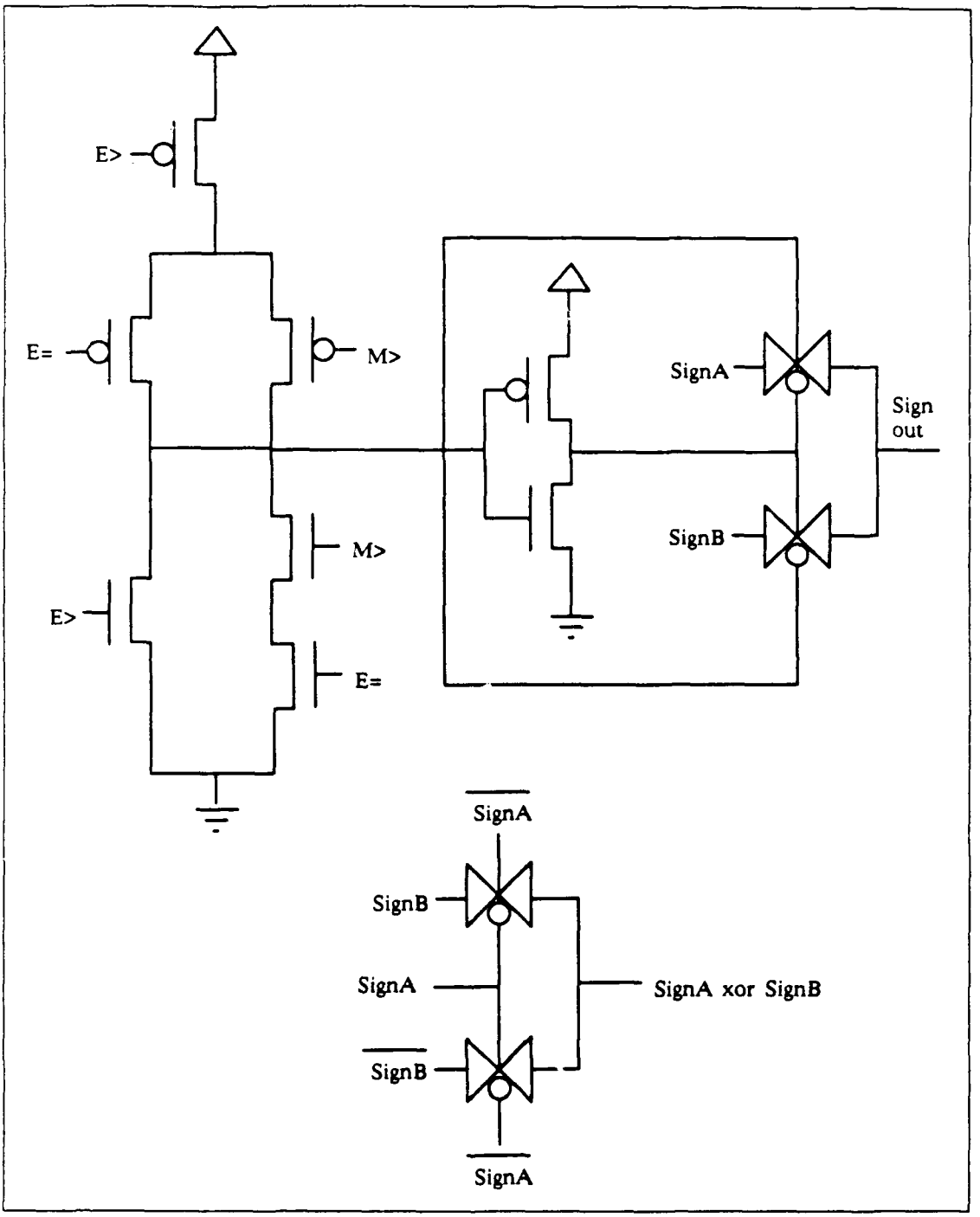

Figure 4.19. Sign Bit Logic

data as the registers for the six least significant bits of the exponent. The registers were added to simplify the data routing. The second adder is a normal carry select adder of three bits driving a three bit carry select. The inverse of the carry-out of $B - A$ is the $E_{A>B}$ signal, which is used as the select control to mux the results of the two subtracters and output the difference between the exponents. The $E_{A>B}$ signal is also used as the select control to mux the two exponents and output the larger of the two exponents (see Figure 4.21). Since the results from the adder cells are only used to calculate $E_{A=B}$, the driver from the sum signal is very small.

The calculations for E_{A0}, E_{B0}, E_{A1}, E_{B1}, and $E_{A=B}$ use the same logical unit (see Figure 4.22). For E_{A0} and E_{B0} the exponent bits are NORed together. For E_{A1} and E_{B1} the exponent bits are NANDed together. For $E_{A=B}$ the resulting bits from the $B - A$ subtracter are NORed together. A pseudo NMOS NOR is used with 11 N-transistors and 1 P-transistor whose gate is tied to ground. The length of the P-transistor gate is modified to produce an output low signal of .5V. The gate length of the P-transistor has been modified instead of modifying the N-transistor's width because this will save area and the delay is not critical for these signals. The gate length for the NOR is 6λ.

Control signals for the mantissa multiplexers have been realized in the exponent area because there is more room and the exponent signals are used to determine the mux controls (see Figures 4.23 - 4.26).

4.4.1.3 Mantissa. The mantissa section is the largest part of the adder unit. The mantissa section determines which mantissa is largest, multiplies the mantissa by two if it is denormalized, places the smallest number on the left processing side, shifts the smaller number's mantissa to match the largest number, determines if an add or a subtract is required, and adds the mantissas together. It consists of eight substructures: the mantissa processing logic, the denormalizing mux, the swap crossbar, the infinity mux, the not-a-number (NAN) mux, the right linear barrel shifter, the subtraction mux, and the final OCM adder (see Figure 4.1).

MANTISSA PROCESSING LOGIC. The mantissa processing logic (MPL) substructure receives the mantissa of A and B from the mantissa section of the main register. It must calculate three signals: M_{A0}, M_{B0}, and $M_{A>B}$. M_{A0} and M_{B0} are determined using the same pseudo nmos NOR gate that is used in the exponent processing logic, except that these gates use 52 N-transistors instead of eleven. The $M_{A>B}$ signal is determined by using a 52 bit subtracter to subtract A from B, the same way that the $E_{A>B}$ signal was determined. A major difference between the two subtracters is that the mantissa subtracter does not require the sum, which decreases the adder cell by 22 transistors per bit (see Figure 4.27). A modified optimized carry-multiplexed adder with

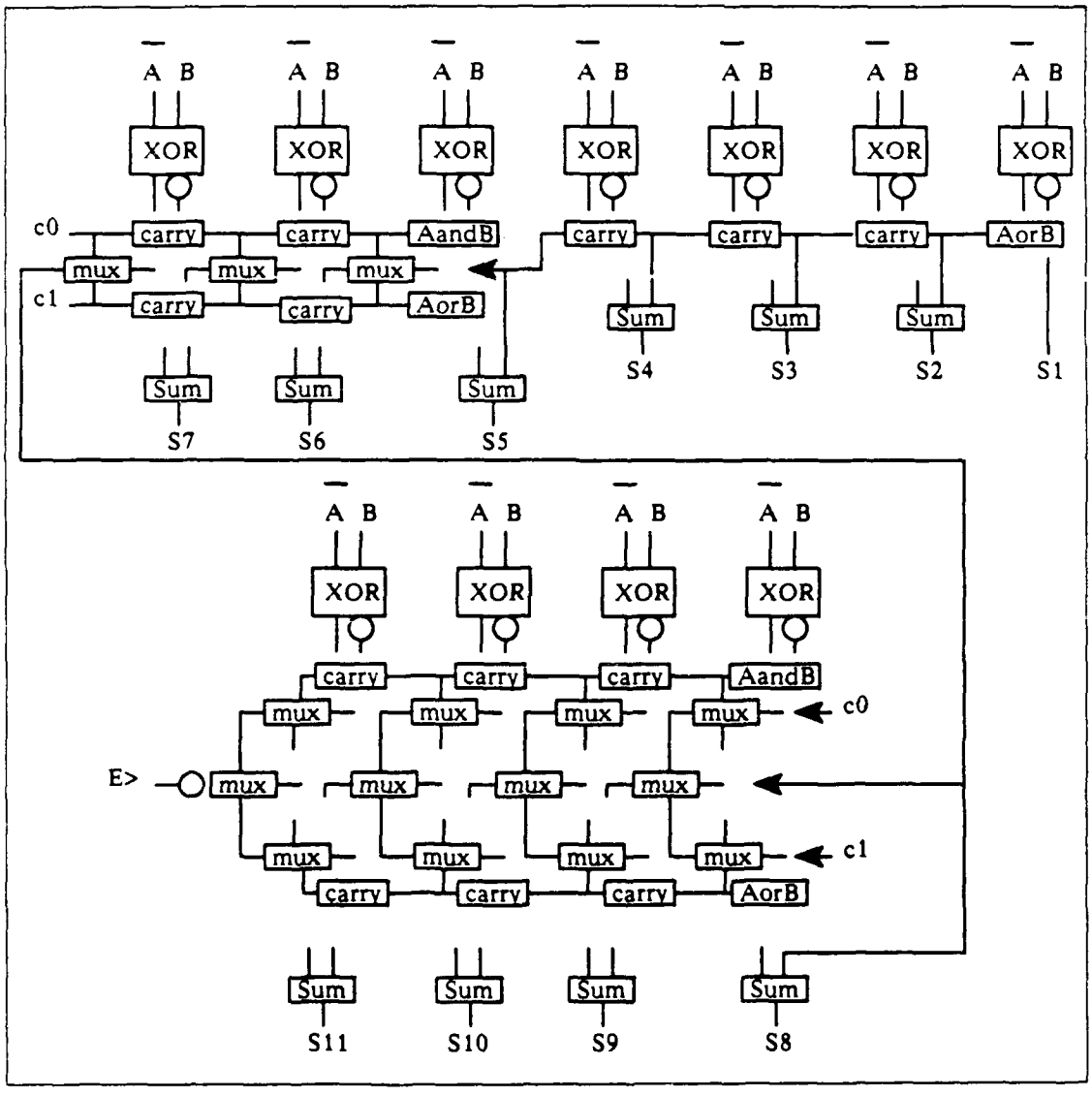

Figure 4.20. Optimized Carry-Multiplexed Subtracter (43,4)

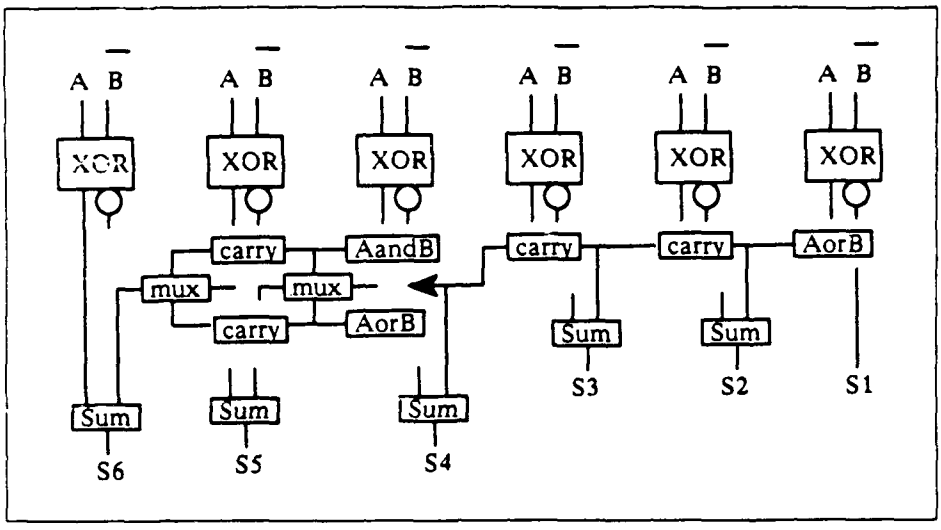

Figure 4.21. Optimized Carry-Multiplexed Subtracter (3,3)

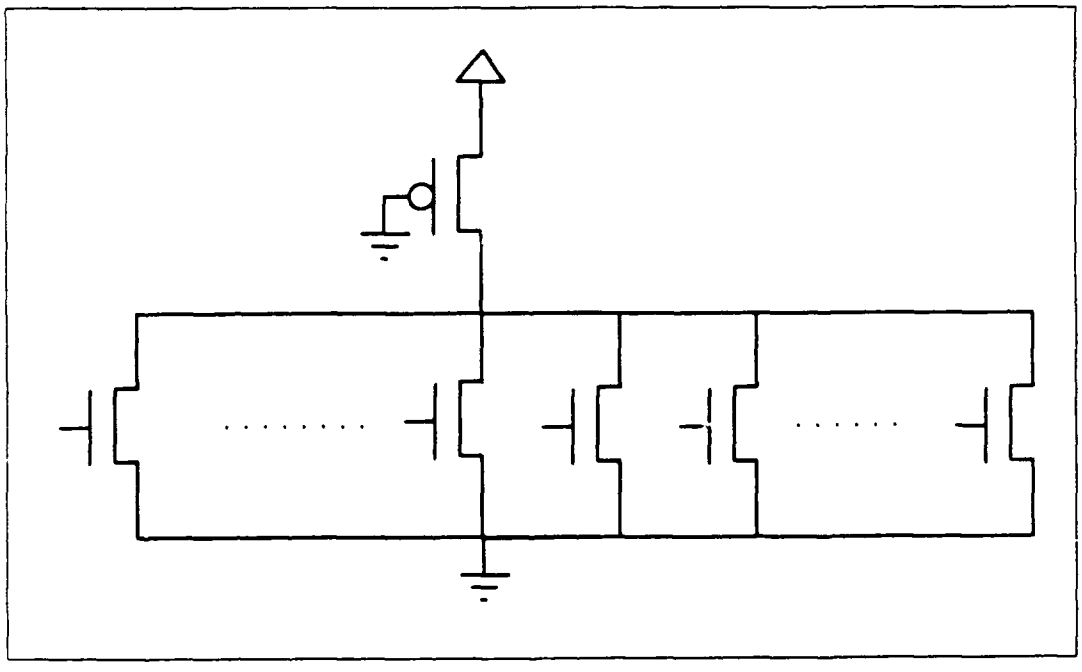

Figure 4.22. Pseudo NMOS Nor Gate

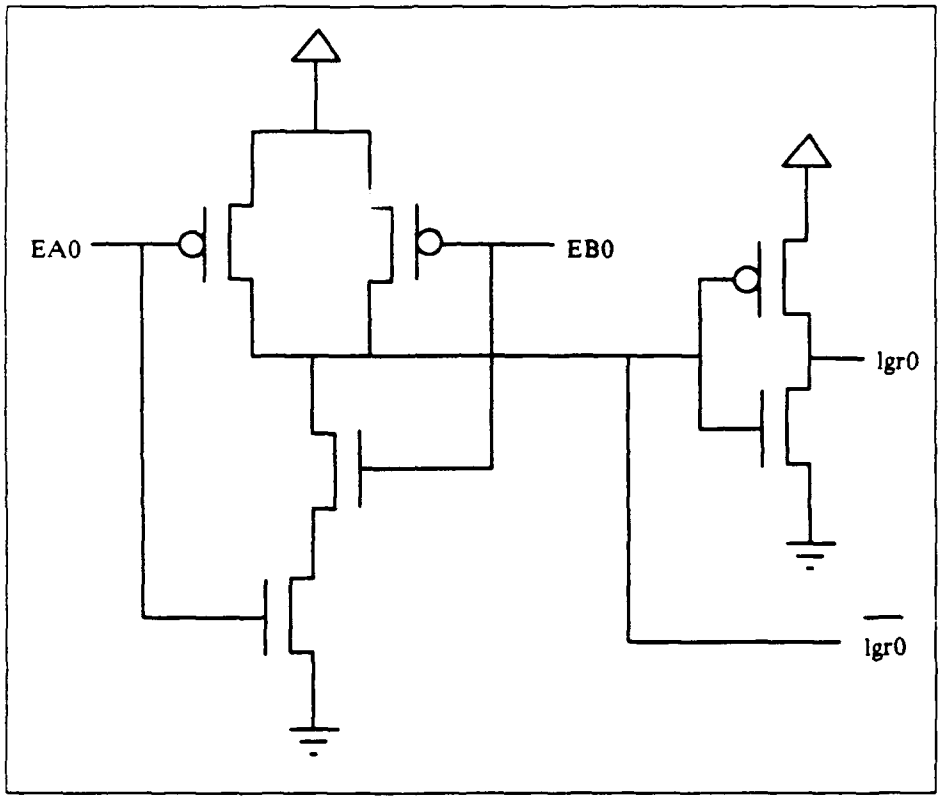

Figure 4.23. larger0

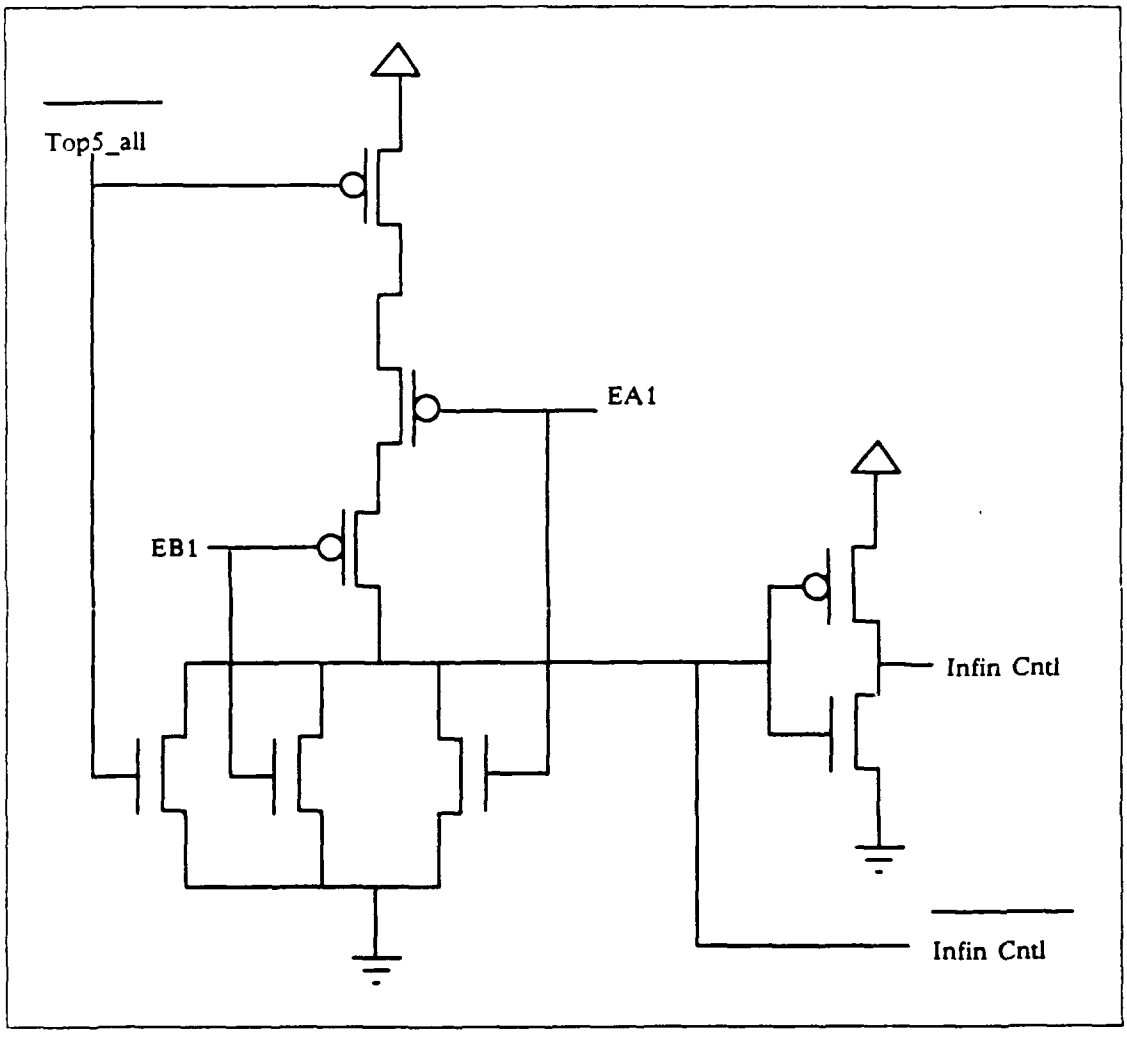

Figure 4.24. Infinity Mux Control

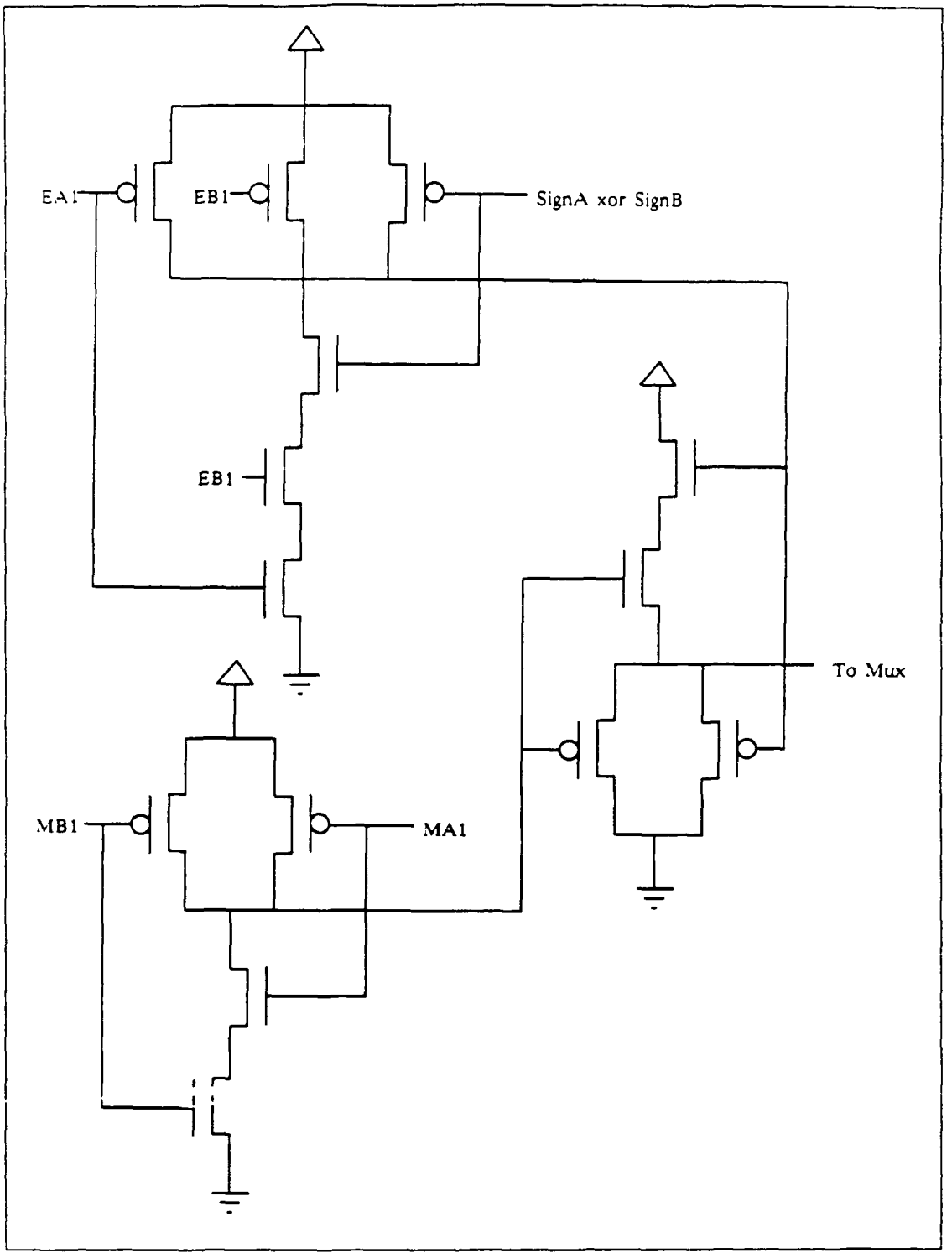

Figure 4.25. NAN Mux Control

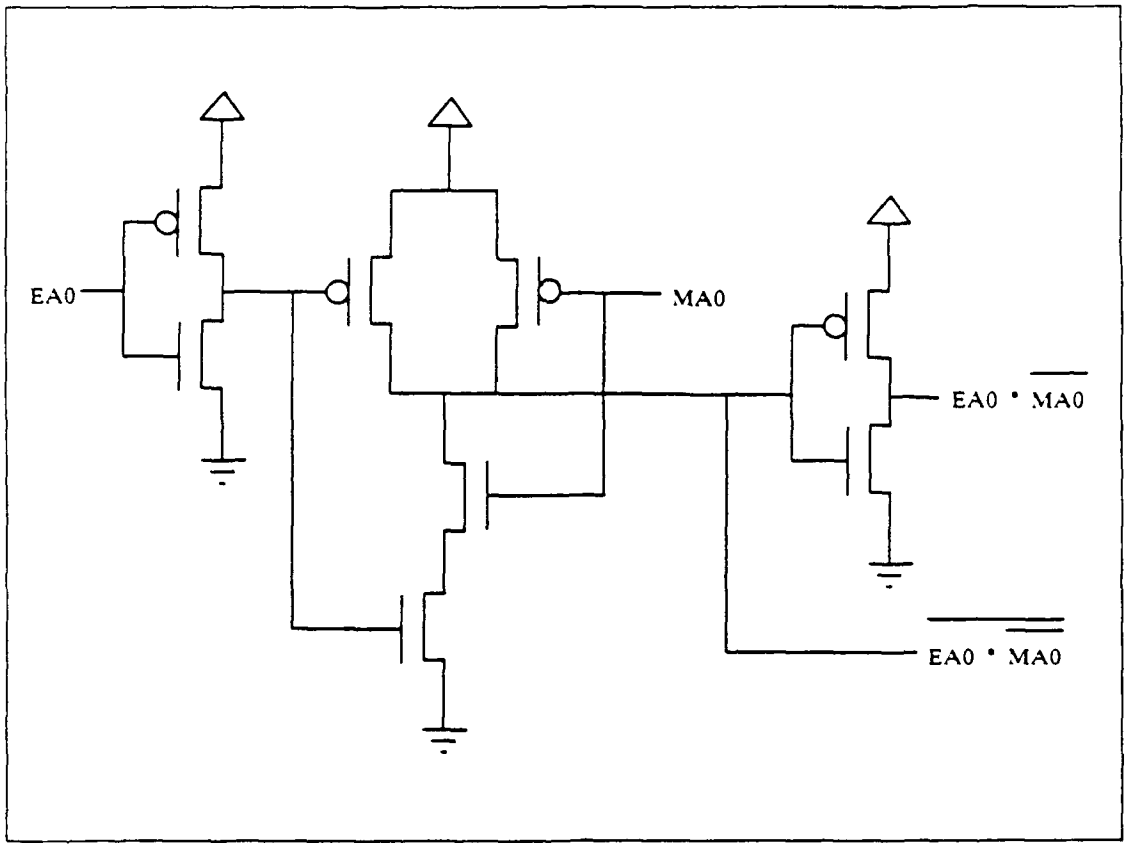

Figure 4.26. Denormalize Mux Control

dimensions (65432, 5432, 432, 321, 3) and an inverted final carry-out is used to calculate $M_{A>B}$. The inputs to the adder are $\overline{A}$, B, and an assumed carry-in of one. The fact that the initial carry-in is one allows the carry-out of the first bit-slice to be realized by $A + B$.

The mantissa comparator uses seventeen different cells in its structure. The cells are much simpler than the regular adder cells. Each cell is 48λ high, while the width varies up to 136λ. The final comparator is 4948λ by 136λ, but about half of the space is empty because the comparator cells do not stack. The empty space is used for the M_{A0} and M_{B0} logic and to route A and B into the denormalize mux.

DENORMALIZE MUX. While $M_{A>B}$ is being calculated, A and B passes through the denormalize multiplexers (see standard mux Figure 4.28). The denormalize mux selects the number with $\overline{Ex0}$ tacked in front (to include the assumed one for a normalized number or a zero for a zero number) or it selects the number with a zero tacked on the end (for a denormalized number). The signal that controls the mux is $E_{A0} * \overline{M_{A0}}$ for the A mux and $E_{B0} * \overline{M_{B0}}$ for the B mux. The denormalize mux can easily operate before the OCM comparator from the MPL substructure can finish, so it is not on the critical path. The output of the mux goes to the swap crossbar.

SWAP CROSSBAR. The swap crossbar places the smallest number on the left bit-slice. It is simply two multiplexers controlled by opposite signals (see Figure 4.29). The crossbar swaps if the exponent of A is larger than the exponent of B or the exponents are equal and the mantissa of A is larger than the mantissa of B. It is controlled by $E_{A>B} + (E_{A=B} * M_{A>B})$. The smaller number leaves the crossbar and enters the infinity mux, while the larger number leaves the crossbar and enters the not-a-number mux.

INFINITY MUX. The infinity mux handles the conditions where the largest number is infinity or not-a-number or the difference between the magnitude of the exponents is greater than 63. Under these conditions the smallest number is set to zero so that it won't alter the infinite number or the not-a-number when the two are added. If either number has an exponent of all ones or the five high order bits of the exponent subtracter are not all one or all zero, then this mux outputs all zeros; otherwise, it passes the number unaltered. The control for the infinity mux is $E_{A1} + E_{B1} + Overshift$. The original architecture for the floating point adder did not contain the logic required to handle the condition for the difference between the exponents. By setting the number to zero, the linear barrel shifter can shift any distance and not alter the number. If the difference between the exponents exceeds 63 then the smaller number is insignificant and won't alter the value of the larger number.

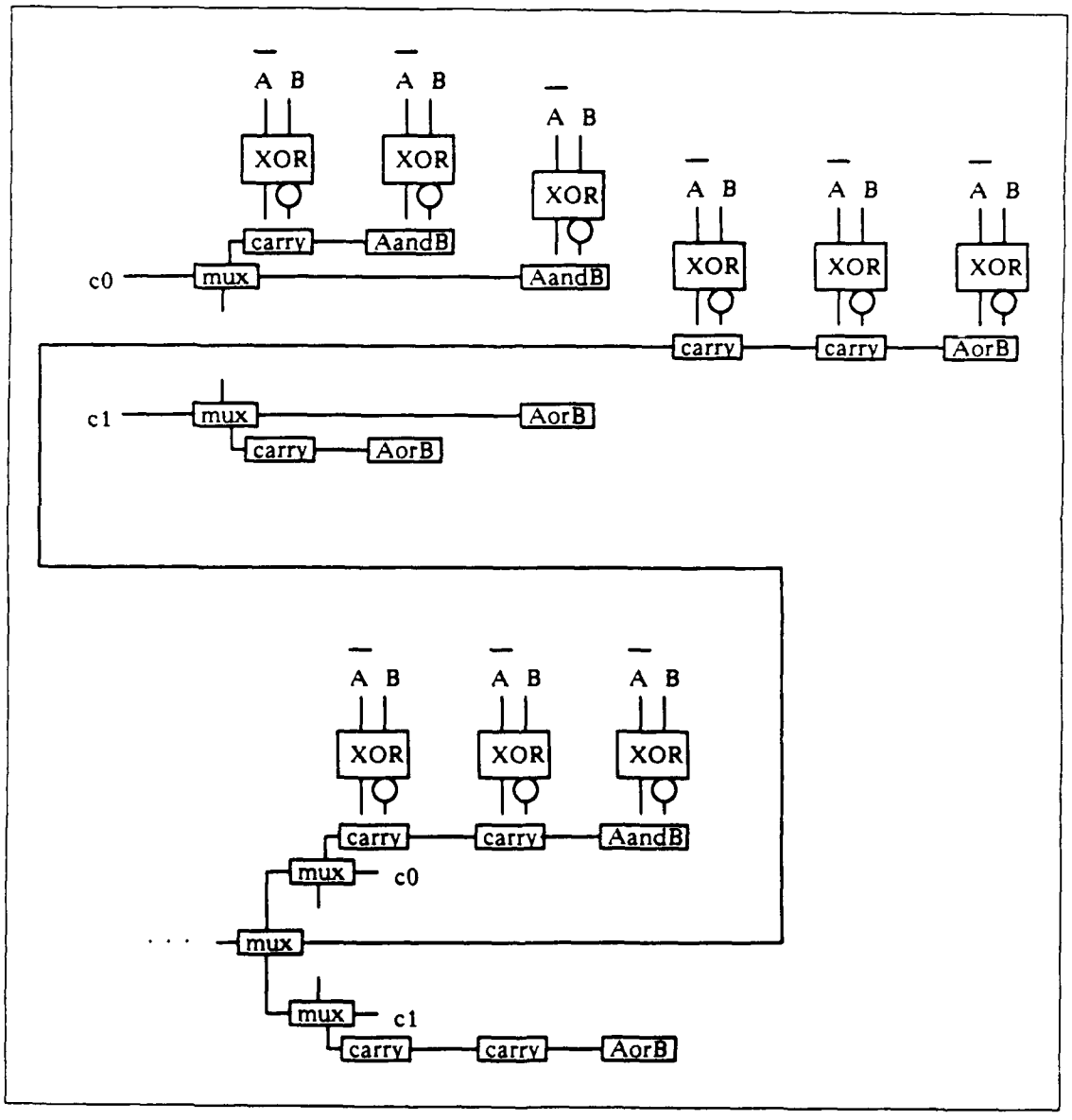

Figure 4.27. OCM Comparitor (321,3)

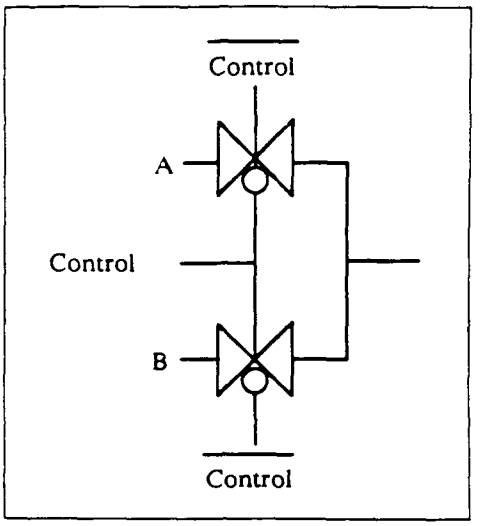

Figure 4.28. 2 to 1 Mux

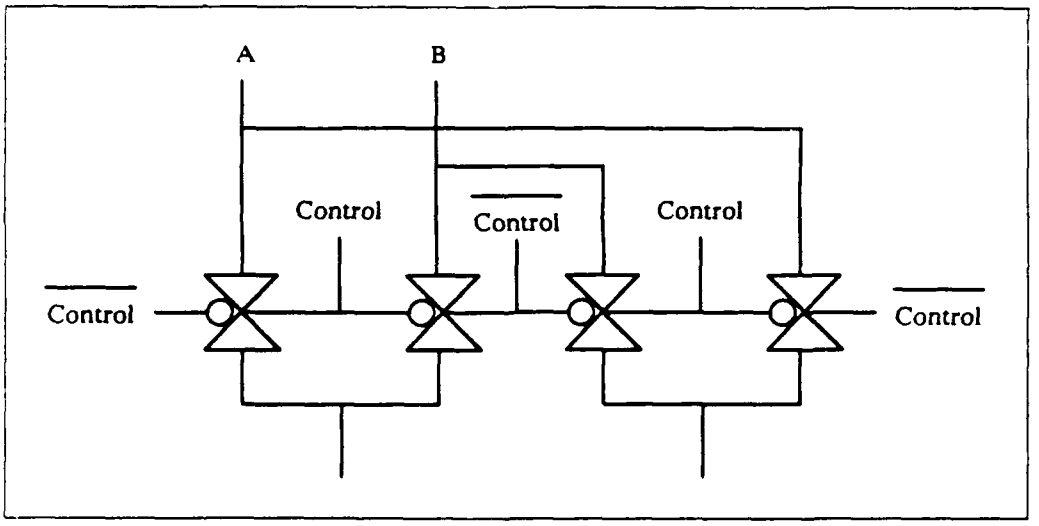

Figure 4.29. 2 x 2 Crossbar

NAN MUX. The not-a-number mux handles the condition of trying to add + and - infinity. It sets the largest number equal to not-a-number if the two numbers, A and B, are both infinite but have opposite signs; otherwise, it passes the number unaltered. Since the mantissa of not-a-number is anything but all zeros, this mux can the reduced down to entering a one in the two high order bits of the number. This implementation requires only two 1-bit muxes and saves 204 transistors from the initial design. The structure created by the three muxes and crossbar is 3926λ by 158λ.

LINEAR BARREL SHIFTER. The right linear barrel shifter receives the output of the infinity mux and logically shifts it the number of bits specified by the difference between the exponents. The controls for the shifter come from the exponent processing logic unit.

The shifter design was supposed to be the easiest structure because the design was going to come from the existing barrel shifter in the LPASP. But, the LPASP barrel shifter is 32 bits wide and takes over 3000λ by 800λ. If the same cells were used then a 53 bit barrel shifter would take too much space. The cells were redesigned by using the N-pass transistor technique used for the earlier muxes. The new design only required about 600λ width, which is acceptable.

SUBTRACTION MUX. The subtraction mux receives its input from the shifter. If the signs of the two numbers are opposite then the mantissas need to be subtracted and the smaller number's mantissa is inverted. The subtraction mux passes the inverted number if the sign bits are different; otherwise, it passes the number unaltered. It is controlled by $SignA \oplus SignB$. The output of the mux goes to the carry select adder. The subtraction mux uses the same mux cell as the other muxes.

FINAL OCM ADDER. The final OCM adder adds the shifted mantissa of the smallest number and the mantissa of the largest number together. If the sign of the two numbers is different then the carry-in is set to one for a subtract; otherwise, it is set to zero for an add. While the smaller mantissa is being shifted, the larger mantissa is buffered up in order to drive the carry chains of the carry select adder. Since the first four bits of the adder have an input of $B = 0$, the XOR unit is not needed. Instead, the A lines are used to control the summation xor and the carry chain xor is replaced with $A * C_{in}$ using a standard AND. The standard AND logic was faster than the T-gate because the output of the gate had to drive another pass transistor. The remainder of the adder is an optimized carry-multiplexed adder with dimensions (654321, 54321, 4321, 321, 21, 2+1). The "2+1" represents a 3-bit ripple half adder that is equivalent to 2 ripple adders because the XOR logic isn't needed. The final carry-out for the adder is not used, so 58 bits are sent on to the priority encoder of the normalize unit.

The mantissa adder uses the same cells as the exponent subtracters except that the sum drivers were larger. The mantissa adder required some additional unique cells because the first four bits of B are zero. For this situation a quarter adder cell was created and some half adder cells without buffers were created. The total number of different cells designed for the mantissa adder is 26. The final structure is 5351λ by 158λ.

4.4.2 Normalize Unit of the Floating Point Adder. The normalize unit takes the summation results from the exponent and mantissa sections of the adder unit and normalizes the resulting number. The normalize unit uses seven different substructures to perform the normalization. They are the result signal generation, an incrementer, an OCM subtracter, a shift control mux, a left linear shifter, an exponent mux, and a mantissa mux.

4.4.2.1 Result Signal Generation. The result signal generation is a substructure used primarily to determine the number of bits that should be shifted to normalize the result of the adder unit. This structure generates two signals. The first signal detected an all zero fraction and is called $Fract0$, while the second signal is the shift number and is called $Ld1Pos$. The $Ld1Pos$ signal is created by a priority encoder.

FRACT0. To detect an all zero fraction, non-standard CMOS logic was used. The logic consists of 58 N transistors in parallel for the pulldown side and a single p transistor on the pull up side. The W/L ratio (n to p) is 4 (designed for worst case when only one n transistor is on). Normally the length is held constant and the width adjusted. However, in this case it was better to adjust the length to keep the size of the transistors reasonable.

LD1POS. The priority encoder was by far the most challenging portion of the normalize unit. Ld1Pos takes 58 inputs and outputs how many bits the mantissa must be shifted. The first plan of attack was to simply use boolean algebra and reduce the function and implement directly. This was not practical for several reasons. First, as the number of input bits increase for a priority encoder the complexity of even the reduced functions grows rapidly. Second, the boolean function reduction tool *expresso* did not work properly. Third, reducing a 58 input boolean function would be extremely error prone.

A better design was created using a tree of nor gates and multiplexors. The operation of the tree is illustrated in this simplified 8 bit example:

> The first 4 bits are checked for all 0 (a NOR operation). If the first 4 bits are 0 then the shift must be at least 4 bits. Therefore the most significant bit of a three bit output

count is a 1. This is exactly what the nor gate outputs for all zero. So the nor result is the most significant bit directly. Next a nor is taken of the first 2 bits and of the fifth and sixth bits. One of these nor results will be the next most significant bit. The msb is used to determine which of the nors will be used. This type of logic continues down to the least significant bit. In this case, though, four nors are used and which nor is used depends on both of the previous results.

The nor portion of the circuit was built mainly with three cells:

twobit a twobit slice with 2 n transistors

twonowire a cell of the same dimensions of twobit –used as a placeholder

ptran a cell with 2 p transistors to complete 2 bit slices

The nor portion of the circuit was designed and constructed by Capt Chuck Wardin as part of his EE695 class project. The signals from the nor portion were used for the selection and control of a sparse matrix of multiplexers. The multiplexers used the same standard mux cell and were arranged as described to generate the correct six $Ld1Pos$ signals.

4.4.2.2 Incrementer. The incrementer is used to increment the resulting exponent that comes from the adder units. The incremented exponent is then available to the exponent mux to be chosen in the case were the most significant bit of the resulting mantissa ($2nd$) is a one. In this case, the mantissa is shifted right one bit and the exponent is incremented.

The incrementer was generated by using some of the half adder cells that were created for the OCM adder. It was configured in a straight ripple half adder design, which is very slow and simple. A ripple adder design was feasible because the exponent result from the adder unit is determined long before the mantissa is determine, which provides plenty of time to settle an 11-bit half adder.

Attached to the incrementer is a detect all ones cell. The detect all ones signal is generated by inverting the output of the incrementer and sending the result through a pseudo NMOS nor gate. The nor gate design is the same as all of the other nor gate designs used to detect all zeroes.

4.4.2.3 OCM Subtracter. The process of normalization requires the manipulation of both the mantissa and the exponent. If the mantissa is shifted to the left by the amount specified by $Ld1Pos$ then the same amount must be subtracted from the exponent. The subtraction is performed by an 11-bit OCM subtracter.

The subtracter is identical to the 11-bit OCM subtracter used in the exponent processing logic of the adder unit, except that the sum outputs have larger drivers. The inputs to the subtracter

are the exponent from the adder unit and the inverted value of $Ld1Pos$. The borrow out of the subtracter is used as an underflow signal to control both the mantissa and exponent muxes.

The output of the subtracter goes through an 11-bit pseudo NMOS nor gate to the exponent mux. The nor gate determines if the output of the subtracter is zero, ($all0$). The $all0$ signal is used as part of the control logic for the mantissa mux.

4.4.2.4 Shift Control Mux. The shift control mux in needed for the condition were the value of $Ld1Pos$ exceeds the value of the exponent coming from the adder unit. In this case, the smaller of the two numbers is used as the shift control and the number is left in the IEEE floating point denormalized format. The underflow signal from the OCM subtracter is used to flag this condition, so it is the control signal to the shift control mux. The inputs to the mux are the six least significant bits of the exponent from the adder unit and the six bits of the $Ld1Pos$. The outputs from the mux are the control signals for the left linear shifter.

4.4.2.5 Left Linear Shifter. The left linear shifter is the structure that shifts the mantissa such that the leading one is just to the left of the decimal point. The shifter uses the same architecture as the right linear shifter, except that the shifted bits are routed to the left instead of being routed to the right. The number of positions that the mantissa is shifted comes from the shift control mux, which is described in the previous section.

4.4.2.6 Exponent Mux. The exponent mux chooses between one of four possible results for the exponent. The mux is designed using three T-gates and an N-pass transistor.

The pass transistor is used to gate in a zero result, which satisfies the conditions where the mantissa is all zeroes or the first and second bits to the left of the decimal of the mantissa are zero and an underflow occurred. The resultant value from the OCM subtracter is selected for the condition where the first and second bits from the decimal of the mantissa are zero and an underflow did not occur. The original value from the adder unit is selected for the condition where the second bit from the decimal of the mantissa is a zero and the first bit from the decimal of the mantissa is a one. The incremented value of the exponent is selected for the condition where the second bit from the decimal of the mantissa is a one.

4.4.2.7 Mantissa Mux. The mantissa mux is used to choose between the five possible values for the mantissa. The mantissa mux is a five-to-one mux that uses four T-gates and one N-pass transistor to gate in a zero result.

The mantissa is set to zero for the condition where the second bit from the decimal is a one, but the incremented exponent results in all ones. An exponent of all ones represents an infinite number if the mantissa is zero. The mantissa is shifted to the right one bit for the conditions where the second bit from the decimal is one and the incremented exponent is not all ones or the second bit from the decimal is zero, the first bit from the decimal is one, and the largest original exponent in the adder unit is zero. The later condition covers the case of adding two denormalized numbers without resulting in a carry. The result from the left linear shifter is selected for the "normal" condition where the second and first bits from the decimal are zero, there is no underflow, and the result from the OCM subtracter is not zero. The result of the left linear shifter is shifted right one bit for the condition where the second and first bits from the decimal are zero and either an underflow occurred or the result from the OCM subtracter is zero. This last condition places the mantissa in the IEEE standard denormalized format. The number must be left as a denormalized number because the exponent is zero and can not be decreased below zero.

4.5 Floating Point Adder Test Chip

A test chip was produced of the floating point (FP) adder on a 84-pin chip. The chip was fabricated using a two micron feature size process. Sixty-four of the chip's pins were bi-directional to accommodate the input of the A and B numbers and the output of the resulting addition. Five input pins were used for control signals. The bi-directional pins were controlled by an enable signal. When *Enable* is low, the pads are configured to input. When *Enable* is high, the pads are configured to output. The A and B numbers are loaded individually by a *LoadA* and a *LoadB* signal. Two clock signals gate the data through the master-slave flip-flops. See Appendix B for the pin out of the floating point adder test chip.

The floating point adder used for the test chip is slightly different from the one that will be in the LPASP. The difference is in the load signal for the master-slave flip-flops (MSFF). The test chip required two load signals, so the MSFFs were modified to accommodate the two load lines instead of the single load line that is required for the LPASP. The extra load signal was added to allow multiplexing of the A and B input data.

4.5.1 Test Results. The FP adder chip was tested on the DAS 9200 logic analyzer. The bi-directional pins were wired to a forcing probe and a sensing probe to allow for both input and output. The remaining control pins were wired to the return-to-zero (RZ) channels of the forcing probes. The RZ channels allow for variations in pulse delay and pulse width so that AC parametric

tests could be performed. Four tests were performed on the 32 test chips to test for shorts between power and ground, chip logic, control AC parametrics, and logic propagation delay.

4.5.1.1 Power-Up Test. The power-up test was performed by setting all of the data inputs low and the MSFF control signals high. This configuration sets all of the chip's transistors to a stable state. The chip power was then slowly increased from 0.0 volts to 5.0 volts and the current through the chip was measured. Two of the 32 chips did not pass this test, but the current of the remaining chips ranged between seven and ten milliamps. The resistance of the two failing chips was 15 ohms and 20 ohms, respectively.

4.5.1.2 Chip Logic Test. The chip's logic was tested by running each chip through a battery of 30 test vectors. The test vectors were chosen to individually test each of the various paths within the FP adder. The adder unit of the FP adder was tested with denormalized numbers, zero, swapping mantissas, not-a-numbers, infinity, exponents differing by greater than 63, mantissa shifting by 1, 2, 4, 8, 16, and 32, and subtraction. The normalize unit of the FP adder was tested for the conditions of an incremented exponent, an unaltered exponent, a shifted mantissa by 1, 2, 4, 8, 16, and 32, a zero mantissa, an exponent underflow, infinity by incrementing of the exponent, two denormalized numbers resulting in a denormalized number, an unaltered mantissa, and a zero exponent.

The logic testing resulted in the discovery of a wiring error made in the left linear shifter of the normalize unit. The gates of the transistors used to shift in trailing zeros for the shift-by-4 and the shift-by-16 rows of the left shifter were wired to the ***noshift*** signal instead of the ***shift*** signal. The result is that when a shift is supposed to occur, the least significant 4 bits or 16 bits (depending on the row) of the mantissa become dynamic nodes. When a shift is not supposed to occur, the least significant bits become tied to ground and their data is lost. Since the least significant four bits are guard bits, only the least significant twelve bits of the resulting mantissa are affected. The error affects the output when the number from the adder unit needs to be normalized; otherwise, the output is perfectly correct. This error will occur in less than 1 out of every 4.2 million possible input combinations.

The logic testing also discovered a fabrication error in two of the remaining 30 chips. The fabrication error seemed to cause the ***Overshift*** signal to be incorrect and it complimented the sign bits of the zero and not-a-number (NAN) results. The latter condition is a logic error but not an output error since the sign bit for zero and NAN is undefined. Combining the results of the two tests gives a fabrication yield of 87.5%.

4.5.1.3 Control AC Parametrics Test. The next set of tests analyzed the AC parametrics of the control signals. The DAS 9200 allows variations in the pulse delay and the pulse width of the RZ channels of the forcing probes. The pulse delay can range from 5 nsec to 99 nsec by increments of 1 nsec. The pulse width can be 5 nsec or range from 10 nsec to 80 nsec by increments of 10 nsecs. The *LoadA* and *LoadB* signals were set with a pulse delay of 5 nsec and a width of 10 nsec. The chip worked with the width of the clock signals set at the minimum of 5 nsec, but the leading edge of $PQ2$ had to trail the leading edge of the load signals by 1 nsec, and the leading edge of $PQ1$ had to trail the leading edge of $PQ2$ by 12 nsec. It was also discovered that the *Enable* signal required 29 nsec to switch the pads from being input pads to being output pads.

4.5.1.4 Logic Propagation Delay Test. The final test of the FP adder analyzed the amount of time required for the data to propagate through the chip. The control signals were set according to Table 4.7. The acquisition sense delay was fixed at 40 nsec, while the $PQ1$ pulse delay was varied. The propagation time was calculated from the time that $PQ1$ was set high to the time that the data pins were sensed.

Signal	Pulse Delay	Pulse Width
LoadA	5 nsec	10 nsec
LoadB	5 nsec	10 nsec
$PQ1$	varied	5 nsec
$PQ2$	6 nsec	5 nsec
Enable	5 nsec	80 nsec

Table 4.7. Control Signal Settings for Propagation Test

Four tests were performed to characterize the propagation time. The first test measured the longest propagation time of the original 30 test vectors without any errors. The fastest chip had a propagation time of 291 nsec. The second test was the same as the first test except that it allowed for one bit to be in error. The result was a propagation time of 272 nsec. The third test ignored the errors produced by the test vector where the mantissa was normalized by 32 bits, and the result was a propagation time of 183 nsec. The fourth test eliminated the seven test vectors that caused normalization of the mantissa to occur. This left 23 test vectors with a propagation time of 132 nsec.

The reason for the improvement in propagation time is that each successive test used less of the faulty left shifter. The final test did not use the left shifter at all and produced the best time. The error in the left shifter is causing the extra delay because its nodes are either dynamic or they are fighting, which extends the amount of time required to settle. The propagation times

were measured using off-chip signal times. The on-chip propagation time can be determined by subtracting the propagation delay of the I/O pads. According to MOSIS documentation, the propagation delay of an I/O pad is about 10 nsec. The resulting on-chip propagation of the FP adder is about 122 nsec. The FP adder used an OCM adder design whose multiplexer signal drivers were not sized properly. An FP adder with properly sized multiplexer drivers and a correct left shifter should have a propagation time of about 90 nsec.

V. Laser Programmable Read-Only Memory

5.1 Introduction

The laser programmable read-only memory (LPROM) is the unit that gives the LPASP its flexibility. The LPROM structure is used in three different sections of the LPASP. The largest section is the ROM area where the microcode is stored. The other two sections are the function ROM and the MAP ROM. This chapter will present the LPROM by giving a historic background of some of the laser applications for integrated circuits followed by a discussion of two laser test chips that were created to learn more about laser applications. The last half of the chapter addresses the components of the newly designed LPROM and the design, fabrication, and testing of the LPROM test chip.

5.2 Background

5.2.1 Introduction. One problem with typical integrated circuit (IC) implementations is that their designs can not be modified after the circuit has been fabricated. Integrated circuits do not contain wires that can be pulled from one location and soldered someplace else. A solution to the problem is to use laser beams to cut or weld circuit wires. Lasers are well suited for cutting or welding ICs because the laser's beam diameter is as small as the connecting lines in ICs. The laser modifications are made during the development and test phase of a chip's design to help detect design flaws or to configure the circuit for some unique task. This background section will focus on how lasers have been used for debugging integrated circuits, restructuring integrated circuits, modifying diodes, hardwiring cell units, and programming read-only memories (ROMs).

5.2.2 Debugging Integrated Circuits. In 1985, a group from the research laboratories of Hitachi Ltd. approached the problem of debugging ICs by using laser beams to cut out bad areas of the circuit(20). They found that .53 micron laser light was absorbed by the top 50 to 100 angstroms of an aluminum layer, which was the same thickness as the aluminum used for IC fabrication. When the aluminum wire was exposed to the laser beam for between 190 to 280 nanoseconds, the aluminum was explosively removed by the high aluminum vapor pressure. If the beam duration was longer than 280 nanoseconds, the silicon layer under the metal line was damaged and the circuit was destroyed.

In order to match the correct laser light and duration, a nitrogen pumped dye laser with a .51 micron wavelength and a pulse duration of 12 nanoseconds was used. The laser had a power density of $.5 \times 10^9$ watts per centimeter squared and was used on 1.75 micron thick aluminum lines.

They found that if the power density was too low, then the metal would just melt and re-solidify, but if it was too high, then the silicon underlayer would be damaged.

They experimented with two types of irradiation methods. The first method, called spot scanning, performed overlapping circular spot scans across the wire. This method was not very effective because it caused damage to the underlayer silicon. The second method, called single shot, used a circular or square spot that was large enough to cut the line with only one pass. They found that the square spot made a cleaner cut than the circular spot, and that the best spot width was between 4.5 and 5.3 microns for a four micron by two micron cut.

5.2.3 Restructuring Integrated Circuits. The researchers at Lincoln Laboratories proposed an alternative approach for modifying integrated circuits with a laser. They wanted to be able to connect structural units together on a wafer after it had been fabricated. By designing a wafer with a large number of unconnected, very common circuit elements, they had the capability to quickly connect up a variety of circuits from the same wafer(14). The idea was to mass produce the wafers and store them on the shelf until one was needed for some specific application. At that time, the wafer connections would be made by the laser, and the development time would be significantly reduced.

To accomplish the connection faculty, a overlapping grid of metal lines were placed on the wafer. Each structural element was connected to a metal line. The metal lines were isolated from each other at their intersections by a layer of silicon. To make a connection, the aluminum-silicon-aluminum sandwich at the intersection was heated with a one millisecond laser pulse from a one to two watt laser. The heat caused the aluminum and silicon to melt together and form an aluminum silicon alloy, which is a good conductor.

They found that the resistance at a heated, or formed, link was less than one ohm, while the resistance at an unformed link was greater than one giga-ohm. The problem with this technology was that the unformed links created significant capacitive loads, which slow down circuit speeds. Lincoln Labs demonstrated the feasibility of using this method by creating a digital integration system used to enhance signal to noise ratios and by creating an array of serial multiply-accumulate cells used to realize a fast Fourier transform processor and a constant false alarm rate (CFAR) filter.

5.2.4 Modifying Diodes. One problem with using lasers on integrated circuits is that the laser leaves craters wherever it scans. The craters make it very easy for the competition to reverse engineer what has been done to the chip. In 1987, two professors at Texas A&M found the solution to this problem and created an additional method of using a laser for IC modifications(11).

They found that the minority-carrier lifetime in silicon could be decreased at least three orders of magnitude if the surface of the silicon were scanned with low powered green laser beams. The laser beam power is so low that it does not create a crater.

The heart of their cell structure was two reverse biased diodes placed between power and ground. The point between the two diodes was connected to the gate of a transistor. When a reverse biased diode is scanned by the laser, its minority-carrier lifetime decreases and its reverse bias current increases. Therefore, if the top diode of the structure was scanned, then the transistor was turned on, and if the bottom diode of the structure was scanned, then the transistor was turned off.

They used the diode cell structure to create a 256-bit programmable read-only memory (PROM). The memory cells have the disadvantage of being larger than the average memory cell because a diode takes more space to create than a transistor. The access time for the PROM was 130 nanoseconds with a 160 nanosecond precharge. The PROM could only handle a maximum frequency of 3.44 megahertz.

5.2.5 Hardwiring Cell Units. The Air Force Institute of Technology (AFIT) has also performed research in the area of laser scanning of integrated circuits. In 1987, Capt Spanburg attempted to use a laser to hardwire a 32-bit comparator and serial multiplexer(18). The idea was to mass produce generic 32-bit comparators and serial multiplexers in the same way that Lincoln Labs planned to do with its wafers. When a comparator was needed, it would be taken off the shelf and hardwired with the laser.

The cell for the comparator was designed using the source of a transistor connected to the input of an exclusive-or (XOR) gate and tied through a long metal line to ground. The drain of the transistor was connected to power. If a high voltage was required as the input to the XOR gate, then the long metal line was cut with the laser. If a low voltage was required, then the cell was left alone.

To accomplish the metal cutting, a neodymium-doped yttrium aluminum garnet (Nd:YAG) laser with wavelength of 1.06 microns and output power of .1 joules was used. The beam diameter for the Nd:YAG laser was 6.35 millimeters with a pulse duration of 20 nanoseconds. The beam needed to be magnified by 635 times to reduce its diameter to 10 microns, but the magnification could not exceed 375 times because the power density of 4×10^6 watts per centimeter squared would exceed the required power density limit for cutting aluminum. Unfortunately, the required optics were not available, and Capt. Spanburg never succeeded in using the laser.

5.2.6 Programming Read-only Memories. Another AFIT student, Capt. Tillie, decided to continue the work started by Capt. Spanburg, but in a slightly different direction. Capt. Tillie's goal was to design a laser programmable read-only memory using a new technology developed at Lincoln Laboratories called diffusion linking(19). For diffusion linking, two areas of silicon are doped next to each other, but separated by 5 microns. When the gap between the two areas is heated by a laser, the dopant will diffuse across the gap and form a conductor. The advantage of diffusion linking is that connections can be made instead of broken, and the overglass is not damaged by the laser (in most cases).

In order to heat the silicon, a argon-ion laser is needed. The laser outputs a 70 to 80 microsecond pulse with a wavelength of 510 nanometers. The optimum power output for diffusion linking is 3.5 watts.

Capt. Tillie was able to incorporate pattern recognition to improve the automation of programming the memory. The pattern recognition was used to precisely align the laser on the chip. Capt. Tillie created a test chip but he was unsuccessful in programming the memory because the precision of the shutter system was not adequate to properly pulse the laser, and he didn't have the correct parameters to control the laser.

5.2.7 Conclusion. Design modifications after an integrated circuit is fabricated can easily be accomplished with the use of a laser. The research labs at Hitachi Ltd. performed a number of experiments that have become the foundation for all subsequent research in the area of metal cutting with lasers. The innovations from Lincoln Labs are very promising, but they can only be used in a limited application where the capacitive loading is not critical. The memory chip created at Texas A&M is an interesting application of solid state physics, but is limited by the large size of the diodes and the slow read access time. The AFIT research is the most discouraging, since a successful application of this technology had not occurred.

5.3 Laser Technology Test Chips

Before any further work could be performed on the laser programmable ROM (LPROM), a design decision had to be made as to whether to use diffusion linking, as Capt Tillie had tried, or to use metal cutting. The solution to the problem required that two test chips be designed and fabricated to compare the two techniques. The first test chip contained simple diffusion linking structures, while the second test chip contained simple metal cutting structures. Both test chips were fabricated through MOSIS. The pads were left completely bare, without static protection or data buffering, in order to make resistance checks across the laser structures.

5.3.1 Diffusion Link Test Chip. The diffusion link test chip contained a wide variety of diffusion link structures, but all of them fell into one of two categories: not containing a transistor or containing a transistor.

The structures that did not contain a transistor were simply a strip of N-diffusion with a three micron gap cut in the middle of the strip. Three different widths for the N-diffusion strip were created to test a four micron link, an eight micron link, and a twelve micron link. One end of the strip was connected to a probe pad, while the other end of the strip was tied to a common metal line that connected a row of these structures to a bonding pad on the side of the chip. All eighteen of the side bonding pads were used to connect rows of the "no-transistor" diffusion link structures. The purpose of the "no-transistor" structures was to test the resistance across the strip after the diffusion gap was linked by a laser.

The structures that did contain a transistor were identical to the "no-transistor" structures except that a polysilicon line was placed across the N-diffusion strip to form a transistor. The polysilicon line was accessible through an additional probe pad so that a voltage could be applied to turn on the transistor. The same three widths were used for the N-diffusion strips as in the "no-transistor" structures, but the gap in the strip was placed at various lengths from the transistor. Groups of the "transistor" structures were connected to the inside edge of the top and bottom row of bonding pads. A modified version of the "transistor" structure, containing probe pads on both sides of the N-diffusion strip, were placed in the center section of the chip between the "no-transistor" rows. The purpose of the "transistor" structures was to see how close a laser link could be placed to a transistor without affecting its performance. The laser link distance from a transistor is a critical issue in a ROM because it affects how dense the ROM can be made.

5.3.2 Metal Cutting Test Chip. The metal cutting test chip is almost identical in design as the diffusion link test chip. The difference between the two test chips is that the metal cutting test chip replaced the N-diffusion strip and its gap with a solid strip of metal. The same three different strip widths were used as in the diffusion strip, and the same two categories of structures were used that either contained or did not contain a transistor. The left half of the chip has "no-transistor" structures that use metal1, while the right half of the chip has "no-transistor" structures that use metal2. All of the "transistor" structures use metal1. Since the metal strip is solid, the laser cut can be made anywhere on the strip to perform the distance test for the transistor.

5.3.3 Lincoln Labs Testing. As was mentioned in a previous section, the AFIT laser workstation that was created by Capt Spanburg and improved by Capt Tillie is not adequate to perform

the delicate work of diffusion linking(19). Fortunately, the researchers at Lincoln Laboratories allowed me to use their facilities to irradiate and test the two laser test chips. The trip to Lincoln Labs was very educational and accomplished the goal of demonstrating the ability to cut metal lines and link diffusion regions with a laser. This section will discuss the equipment used to perform the cutting and linking, present some of the problems that have been discovered, explain the tests that were performed on the two tiny chips, and conclude with a recommendation on which method should be used for the LPROM design.

5.9.3.1 Equipment. Lincoln Labs uses slightly different equipment then the AFIT laser workstation. It is capable of making seven links or cuts per second. They use an argon ion laser for linking diffusion and cutting metal. The only differences that they make between the two processes is the amount of power that is used. The linking process uses 2 - 3 times more power than cutting. The power for the laser is completely computer controlled, so diffusion linking and metal cutting can occur in the same process run.

They use the same optics box as AFIT, except that they replaced the illumination lamp with a fiber optics line and took out the beam size adjuster. The lamp was replaced because they found that the aluminum head on the optics box was very sensitive to the heat that the lamp produced. To change the beam size, they move the platform in the z-axis to place the beam out of focus. They adjust the size on the upper part of the conic section of the focused beam until the diameter is correct. After fixing the beam size, they focus the camera and then use a z-axis translation stage to adjust variations in the height of the chip. They have also added on optics platform on top of the z-axis stage. The platform allows for adjustments in pitch, yaw, and rotation in order to get the chip perfectly flat and squared on the translation table.

The researchers at Lincoln Labs have created a device to calculate the size of the beam diameter. They use a diode with a slab of metal partially covering it and some filters to decrease the intensity of the beam. They step the beam across the diode and measure the current that is created. As the beam moves onto the metal slab, the current decreases to zero. The results are fit to a Gaussian curve and the diameter is determined from the curve.

5.9.3.2 Problems. There have been some problems discovered in the process of perfecting diffusion linking and metal cutting. The most irritating problem is that the optimum .neters for the process vary depending on the fabrication run. They have found that even different fabrication runs by the same vender produce different process parameters. In the words of Matt Rhodes, the design engineer, "this is not a cookbook process"(15). To solve this problem,

they always make a set of test chips on the same fabrication run as the real chip that they are producing. They have found that fabrication vendors that use lightly doped drains and a nitrite based passivation layer that absorbs the laser light create chips that are difficult to diffusion link. When the passivation layer absorbs the light, it melts at a high temperature and pulls away from the laser beam. The molten layer can then come in contact with a metal layer, which melts the metal and can cause an open circuit. Another problem that they have found with diffusion linking is that the reversed diode leakage at the diffusion link increases by an order of magnitude. This problem wouldn't effect our LPROM design, but it should be kept in mind for future projects. Since the diffusion linking alters the substrate, they recommend as a rule of thumb to keep a 5 - 7 micron distance from the linking area to any other device on the chip.

5.3.3.3 Tiny Chip Tests. The metal cutting tiny chip used a spot size of three microns. The metal1 layer was cut using an 84 microsecond pulse and .6 to 2.0 watts of power. The power was varied to see how sensitive the process was to power variations. The metal2 layers that had glass cuts were also cut with an 84 microsecond pulse, but the metal2 lines with a passivation layer were cut using a one millisecond pulse. The longer pulse was needed because the metal2 lines were wider. The probing unit was not available, but visual inspection verified that the cuts were made successfully.

The diffusion link chip used a spot size of three microns and an 84 microsecond pulse. The power was varied between 2.2 and 3.2 watts. Some of the links were probed. The passivation cut versions did not seem to work, but the links with passivation worked will. The resistance varied between 108 to 280 ohms. The probe pads on the chip were too small to get a good reading, so the passivation cut versions may have linked, but just not been probed well.

5.3.3.4 Conclusions. The trip helped to understand that the diffusion link process is a reliable method, once the correct parameters are found, of linking signals together on a chip. There are some advantages, in this specific application, to the metal cutting process over the diffusion link process. The resistance of the diffusion link is greater than the resistance of a metal line. This is somewhat important for our application, but not critical because the link is in series with a transistor which has a larger resistance. The capacitance of the diffusion link is greater than a metal line. This is because the diffusion link acts as a reversed bias diode between the diffusion and the substrate. One possible solution to the capacitance problem is to enclose the link in a separate p-tub with a negative voltage, for example -3 volts. The negative voltage will widen the depletion region and decrease the capacitance, but also increases the area by 10%. The final advantage of metal

cutting is that it will allow testing of the LPROM transistors before the LPROM is programmed. This test will be important to verify the quality of the memory cells.

5.4 Components of the LPROM

The advantages of the metal cutting technique outweigh the disadvantages, so it was decided to design a new LPROM using the metal cutting technique. The main restriction on the new LPROM was that it must work with the existing XROM. Because of the XROM restriction, the new design started with the XROM design and modified it to be suitable for the metal cutting technique. From discussions with Matt Rhodes, it was determined that the length of the metal line to be cut should be a minimum of six microns, and that parallel metal lines should be at least three micron from the cut.

The resulting LPROM cell stores four bits in a 35λ by 26λ area, which gives a density of $227.5\frac{\lambda^2}{bit}$ (see Figure 5.1). The new LPROM cell is about one-fourth the density of the XROM, but it is about twice as dense as the diffusion link LPROM. The new LPROM uses the same supporting cells as the XROM, but only uses half of the bit lines. The remainder of this section will discuss the components of the complete LPROM (see Figure 5.2).

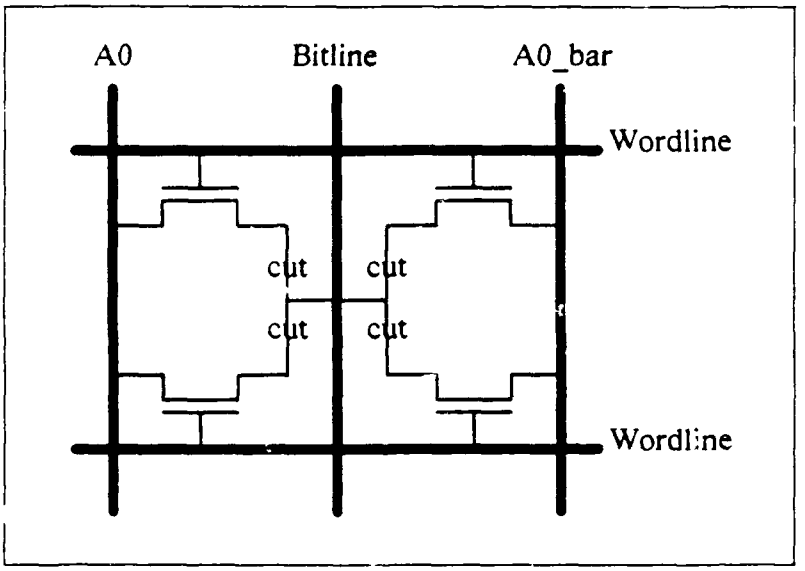

Figure 5.1. LPROM Cell

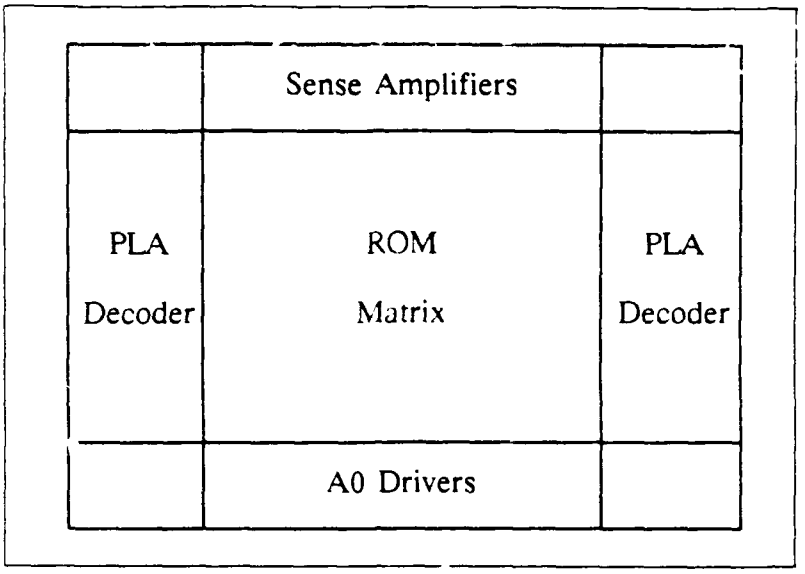

Figure 5.2. LPROM Block Diagram

5.4.1 A_0 Drivers. The bottom of the LPROM is composed of two almost identical cells that alternate across the bottom. The purpose of the cells is to buffer the least significant address line, A_0, and its inverse, $\overline{A_0}$. Both of these signals are sent up the entire length of the ROM matrix. The signals alternate horizontally across the ROM matrix. The LPROM version of the driver cells has been modified slightly from the XROM version to compensate for the decreased density. The only modification was to cut in half the number of connections to the A_0 and $\overline{A_0}$ lines in the ROM matrix.

5.4.2 Sense Amplifiers. The top of the LPROM is filled with sense amplifiers that tie in the bitlines from the ROM matrix. The lower half of the sense amplifier contains the precharge logic and a four-to-one multiplexer that receives four of the bitlines from the ROM matrix and chooses one of them to be sent to the upper half. The four control signals for the multiplexer are generated by a two-to-four demultiplexer that receives its input from the second and third address lines, A_1 and A_2. Since the LPROM has half the horizontal density, the A_2 line is tied low. The only modification made to the lower half was to help route the bitlines to the correct multiplexer input.

The upper half of the sense amplifier cell contains the sense amplifier and the logic to invert the bitline data, a technique used to decrease capacitance on the bitline of the XROM. For the LPROM, the word sign signal is tied low so that the bitline is not inverted. The word sign signal

should be watched carefully. If the XROM compiler ties the word sign high then the LPROM data representation associated with the corresponding bitline must be inverted. One approach would simply be to tell the XROM compiler NOT to change the word sign. The upper half of the sense amplifier cell was not modified.

5.4.3 PLA Address Selectors. The remaining upper address lines, A_3 to A_x, are sent through an array of PLA address selectors to select and drive the ROM matrix wordlines. In the case of the LPROM, the A_2 line is not used in the two-to-four demultiplexer, so the upper address lines range from A_2 to A_x. The address PLAs are located on both sides of the ROM matrix. The PLAs on the left side of the ROM matrix select the upper wordlines and shunt the lower wordlines. The PLAs on the right side of the ROM matrix select the lower wordlines and shunt the upper wordlines. Both sides use the same cell array, but one side is mirrored and vertically shifted from the other side. The shunts are used to decrease the resistance of the wordlines by shunting parallel metal2 and polysilicon lines on both sides of the ROM matrix. The LPROM PLA cells were stretched vertically from the XROM versions to accommodate the decreased vertical density. The horizontal dimension of the cell did not change.

5.4.4 ROM Matrix. The ROM matrix is a two-dimensional array of LPROM cells. Each LPROM cell contains four transistors and five signals. The three signals that run vertically are A_0, bitline, and $\overline{A_0}$. The bitline signal is in the center. The remaining two signals are the upper and lower wordline signals that run horizontally. The transistors are located in the four corners of the cell. All of them are connected to the bitline on one side and either the A_0 or the $\overline{A_0}$ line on the other side. The upper transistors are gated by the upper wordline, and the lower transistors are gated by the lower wordline.

5.4.5 Functional Description. The LPROM works in two phases. The first phase precharges the bitlines high while the address lines are settling. The bitlines are precharged with a precharge signal that gates V_{dd} through an N-pass transistor to the bitline. The N-pass transistor is used to keep the precharge voltage at about 3.6 volts. The second phase occurs when the precharge signal goes low and the wordlines connect the bitline to the A_0 or the $\overline{A_0}$ line through the ROM transistors. If the A_0 address is high then the A_0 column signal is driven low. If the connection to the A_0 line has not been cut and the A_0 line is low, then the charge on the bitline will drop. The dropping charge is detected by the sense amplifier and a high signal is output. If the connection to the A_0 is cut then the bitline will remain high and the sense amplifier will output a low signal.

Therefore, the presence of a transistor connection to the bitline produces a high output, and a laser cut of the connection will produce a low output.

5.5 LPROM Test Chip

A test chip was produced of the LPROM on a 40-pin tiny chip. Due to the size and pin limitation of the tiny chip, the LPROM was limited to 256 words of 16 bits each. The chip requires eight address lines and a precharge line. It outputs sixteen data lines. The lines are not multiplexed and do not need to be enabled as in the floating point adder test chip. See Appendix C for the pin out of the LPROM test chip. In addition to the normal MOSIS fabrication, a ground line was dropped to the chip cavity to more evenly distribute the electric potential of the substrate. The ground line is used because the ROM matrix does not contain any substrate contacts.

The PLAs of the LPROM were arranged so that the wordlines start with 000000xx at the top and work consecutively down to 111111xx. If the five most significant address bits are observed, then the even numbers are on the left side and the odd numbers are on the right side. The A_0 bit is used to discharge the bitline, and the A_1 bit is used to select the correct bitline.

The A_0 column line and the $\overline{A_0}$ column line alternate across the ROM matrix starting with the A_0 column line on the leftmost side of the ROM matrix. Because the signals alternate, the address selection of the two least significant bits going from left to right across the ROM matrix are convoluted such that the addressing is 1, 0, 2, 3.

A computer program was written to convert binary data into X,Y coordinate laser cut locations for the laser workstation at Lincoln Labs. The program takes the address convolution into account and also calculates the coordinates so that the laser takes the most efficient route. During the laser programming of the LPROM, the laser will work its way around the LPROM in a snake-like motion instead of a typewriter motion, which is inefficient.

5.5.1 Preprogram Testing of the LPROM. Two tests were performed on the LPROM chips before they were sent to Lincoln Labs to be programmed. The first test checked the current through the chip with all of its inputs low. The current ranged between 11.1 and 12.4 milliamps, which was acceptable. The second test ran through every address and verified that the bitline was being discharged through the transistor. The second test used a word generator to produce the precharge signal and the five most significant bits. The remaining three least significant bits were wired to three manual switches. A logic analyzer was used to read each output bit and verify that the

voltage was high when the precharge line was low. All four of the LPROM test chips passed this test for every address at every output.

5.5.2 Postprogram Testing of the LPROM. The LPROM chips were programmed such that the upper eight bits of data were identical to the lower eight bits of data. With eight bits of data and eight address lines, every possible combination of eight bit numbers could be programmed on each chip. Two of the chips were programmed so that the output data was sequential from 00 hex to FF hex. The other two chips alternated low and high values to make the laser cuts more distributed. The output data for these chips was as follows: 00 FF 01 FE 02 FD etc. Each chip contains 4096 bits of data, and each chip had 2048 laser cuts. Each bitline on the chip contained the same number of cut transistors, but not in the same pattern of cuts.

Testing for the LPROM chip was accomplished using the DAS 9200 logic analyzer. The address pins were wired to a forcing probe, the data pins were wired to two sensing probes, and the precharge pin was wired to the return-to-zero (RZ) channel of a forcing probe. Two test were perform on the chips. The first test verified the accuracy of the laser programming. All of the chips were programmed accurately and the chip logic worked perfectly. The yield from the laser programming was 100%.

The second test analyzed the AC parametrics of the chips. The precharge signal was tested first by decreasing its pulse width and its pulse delay. The pulse width and delay were decreased down to 5 nsec, which is the minimum allowed for a RZ channel, and the chips still functioned properly. Since the width and delay could not be decreased any shorter, the actual setup time for the LPROM chips could not be determined. The second parametric tests measured the off-chip read access time. The read access time was taken to be the time from the precharge signal going low to valid data on the output pins. The tests used a precharge delay of 5 nsec and various pulse widths. The results of the test are shown in Table 5.1.

The tests concluded that the precharge width has little effect on the read access time. The shortest off-chip read access time was 23 nsec. The on-chip read time can be determined by subtracting the propagation delay associated with the pads. MOSIS tiny chip pads were used which have a documented propagation delay of between 8 to 11 nsec. The resulting propagation delay for the LPROM is between 12 to 15 nsec.

Chip	Precharge Width	Read Access Time
A	5 nsec	25 nsec
B	5 nsec	27 nsec
C	5 nsec	25 nsec
D	5 nsec	26 nsec
A	10 nsec	No Data
B	10 nsec	27 nsec
C	10 nsec	25 nsec
D	10 nsec	23 nsec
A	60 nsec	No Data
B	60 nsec	28 nsec
C	60 nsec	25 nsec
D	60 nsec	31 nsec

Table 5.1. LPROM Read Access Times

VI. Parallel Applications

6.1 Introduction

The architecture of the LPASP has been upgraded a number of times to include such things as assembler language support, for instance. The purpose of this chapter is to investigate the additional hardware and software requirements to support parallel processing of multiple LPASPs.

Parallel processing can be accomplished using a variety of techniques. The best technique, though, usually depends on the expected application for the processor system and the facilities available on the processor. The chapter begins with a presentation of various options that are available to a parallel processing architecture. The options section is broken into four subtopics which includes processor communication, memory architectures, parallel verses pipelined architectures, and application mapping. The second section recommends which approach the author feels is best to support the Kalman filter algorithm for the LPASP. It also addresses the hardware and software requirements for the suggested architecture.

6.2 Options

There are numerous techniques available to support parallel processing. Each technique has its own advantages and disadvantages depending on the application. The intent of this section is to give the reader a feel for what questions need to be answered before an architecture can be developed. This section will explore four of the major issues associated with parallel processing. The first issue is how the parallel processors will communicate with each other. The second issue is how the memory will be managed and allocated. The third issue is whether to place the multiple processors in a parallel architecture or a pipelined architecture. The final issue discussed is the mapping of the algorithm to the parallel architecture.

6.2.1 Processor Communication. Processors communicate with each other through what is called a network. A network can pass information from one processor to another in a variety of ways. This section will discuss the four design decisions that must be made to select an appropriate network architecture.

In the world of parallel processing, people are continuously debating which network architecture is the best. The answer to this burning question is, "It depends". The real issue is what questions need to be answered in order to determine the best network for your application. The four design areas that need to be addressed are the operating mode, the control strategy, the switching methodology, and the network topology(5).

The operating mode determines whether the network will be synchronous or asynchronous. If the network communication paths are established in a synchronous (lock-step) fashion then the network is said to be synchronous. If the individual paths are dynamically established then the network is said to be asynchronous. Synchronous networks have a simpler design and are easier to control. They can be effectively utilized when processor communication occurs all at once with a nonconflicting, one-to-one source to destination correspondence. This type of communication is found in a Single Instruction / Multiple Datapath (SIMD) architecture. The disadvantage of a synchronous network is that it is very restrictive and doesn't have the flexibility of an asynchronous network. The speed of the synchronous network is limited by the slowest element.

The control strategy is the method of controlling the switching elements and the interconnecting links that make up the network. The two general control strategies are centralized control and distributed control. In a centralized control network, a central controller commands all of the switching elements and the interconnecting links. The central controller performs all of the information collecting and voting required to decide how the network should be configured for each communication cycle. The centralized control strategy allows the network to see the big picture and choose the optimal configuration, but it requires a lot of sensor and control wiring, and the central controller's configuration algorithm can be rather complex for large networks.

The distributed control strategy allows the individual elements of the network to determine their own configuration. Each element of the network examines the destination address of its input data and uses the information to determine its configuration. The advantage of the distributed strategy is that elaborate wiring and complicated control algorithms can be avoided. The disadvantage is that the overall network configurations is not necessarily optimal.

The third decision that must be made is the switching methodology. The switching methodology is the way that a piece of data traverses its way through the network. There are two methods of switching through the network. The older and less commonly used method is circuit switching. Circuit switching provides a dedicated path through the network connecting the source and the destination prior to the transmission of data. The transmission path is held until the data transmission is completed. This switching method is similar to the old telephone switching network. The advantage of circuit switching is that it is very simple. The disadvantage is that the dedicated paths can cause network congestion.

The more commonly used switching methodology is packet switching. With packet switching, the data is broken in packets that contain the routing information. The packet(s) are then sent individually through the network. The packet switching methodology is similar to the postal

system. Since the packets do not require dedicated lines, this methodology allows network resources to be shared, which reduces congestion. The disadvantage of packet switching is that the network configuration and control can become complex and the data packets can be misrouted and lost.

The last major decision is the network topology. The network topology is the flexibility of the links within a network. Two types of network topologies are static networks and dynamic networks. The links within a static network are passive, dedicated busses. However, the links within a dynamic network are reconfigurable via control structures. Dynamic networks are more flexible than static networks, but dynamic networks require more control logic than static networks.

The network topology and the operating mode sound similar, but they are different. The operating mode is how the network will be used, while the network topology is the flexibility of the network. As an example, a network with static topology can operate in an asynchronous mode. In this example, static links would be established and released at random to accommodate asynchronous communication.

6.2.2 Memory Architectures. The memory architecture of a processor system has a big impact on how the interconnection network is used. There are two types of architectures that are commonly used in parallel systems(5). The first type is a processor-to-memory architecture. The second type is a processor element (PE)-to-PE architecture. The processor-to-memory architecture uses a central memory system. The PE-to-PE architecture uses a distributed memory system.

6.2.2.1 Processor-to-Memory. In a processor-to-memory architecture, the parallel processors are connected to the memory through the interconnection network (see Figure 6.1). A bi-directional network is used to perform all of the communications between the processors and the memory units. Interprocessor communication is performed by storing and retrieving the communication information in memory units that are common to the two processors that are communicating.

The centrally located memory system of the processor-to-memory architecture makes interprocessor communication very easy, but it also increases the memory access time because of the extra time required by the interconnection network. The memory accesses are accomplished by sending out a memory request into the network. When the appropriate memory unit receives the request, it returns a block of data to the requesting processor. Instruction fetches are performed the same way as any other memory request, which can heavily load the network. Instruction fetches should not be frequently required for the LPASP, though, because most of its instructions will be store in the XROM or the LPROM.

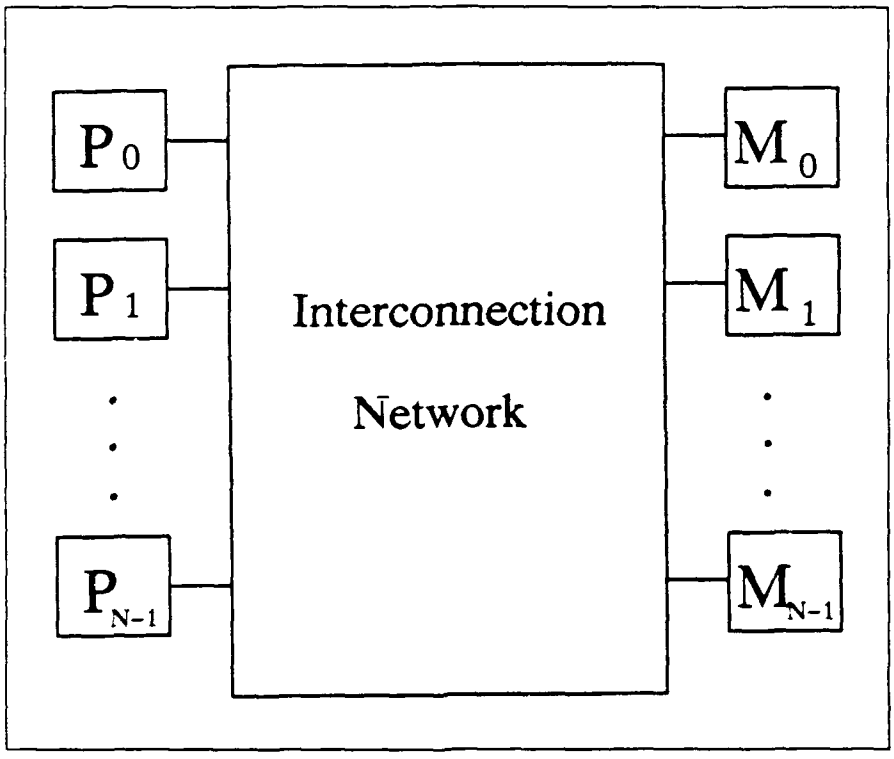

Figure 6.1. Processor-to-Memory Architecture

6.2.2.2 PE-to-PE. The PE-to-PE architecture uses processor elements that consist of a processor and a local memory unit paired together. The interconnection network is uni-directional with a bank of input ports and a bank of output ports. Each processor element has an input port and an output port that is connected to the respective output and input ports of the network (see Figure 6.2). Unlike the processor-to-memory architecture, this architecture requires a one-to-one correspondence between the number of processors and the number of memory units, and the network only provides interprocessor communication. Interprocessor communication is initiated by a processor element placing data onto an input port of the network. With the proper configuration of the network, the data is driven out of the network output port and into the input port of the destination processor element.

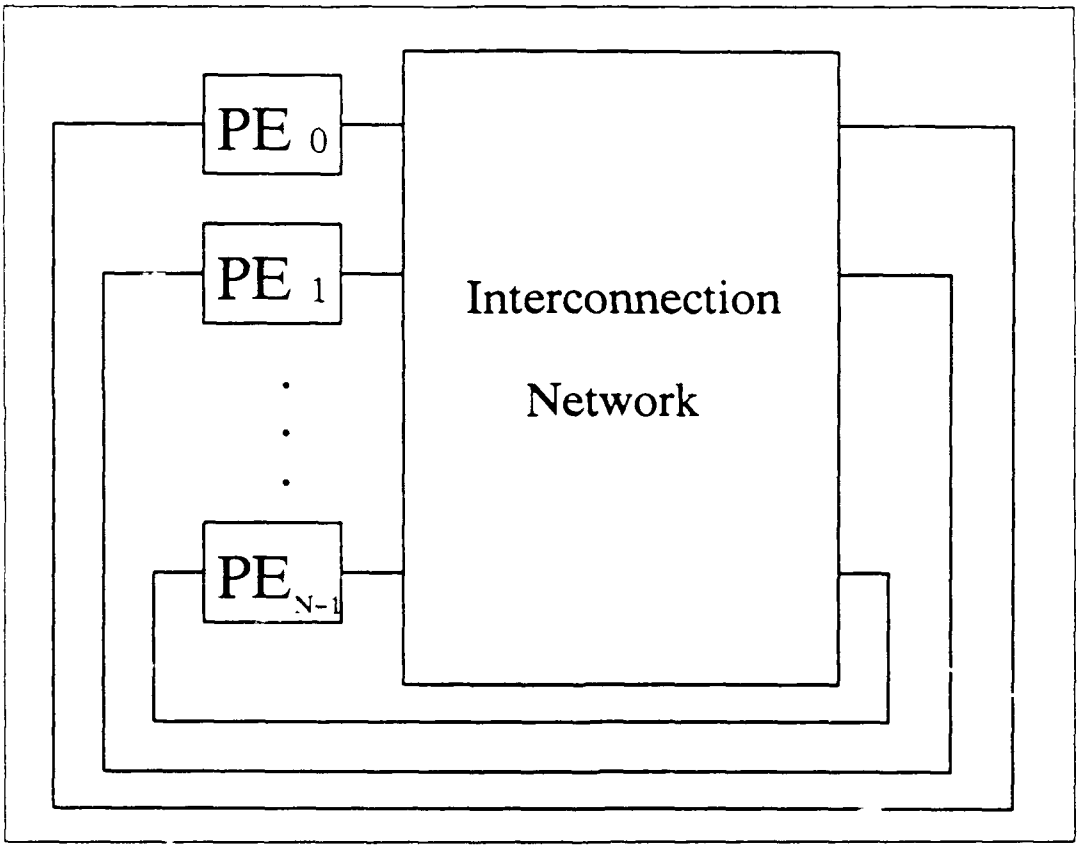

Figure 6.2. PE-to-PE Architecture

The memory unit within a processor element is connected directly to its associated processor, so memory accesses do not require the interconnection network. Since the network is not needed for memory accesses, memory access times can be kept to a minimum and the load on the network

is reduced. The distributed memory structure causes interprocessor communication to be more complex and slower than normal memory accesses. Since the memory is not centralized, if a processor needs information that is only in another processor element's memory then an interprocessor transfer must occur to obtain the information.

6.2.3 Parallel vs. Pipeline. Pipelining, also known as temporal parallelism, is another way to increase the performance of a processing system. This section will first compare and contrast pipelined systems verses parallel systems, and then explain how the LPASP could be used in a pipelined architecture.

6.2.3.1 Compare and Contrast. Both pipelined and parallel computers attempt to increase performance by increasing the number of simultaneous operations being performed. Pipelined computers accomplish the increase in performance by partitioning a function into subfunctions that can be executed concurrently. Parallel computers increase performance by replicating hardware to perform independent execution of subprograms on separate hardware. Pipelined computers measure their performance as one result every $\frac{T}{N}$ seconds, while parallel machines measure performance in terms of N results every T seconds(5). Performance for the pipelined machines is best seen when processing one-dimensional vectors of arbitrary length, while array problems with lengths that are multiples of the number of processors are best executed on a parallel machine. Both systems require high speed memory systems, but pipelined systems use a single, multi-way, interleaved memory and parallel systems use multiple, independent memory modules. Another distinction between the two machines is that detailed control is handled largely by the hardware in a pipelined machine, while it is handled by the user or the operating system in a parallel structure.

6.2.3.2 Pipelining with the LPASP. Pipelining can be accomplished with the LPASP by cascading multiple LPASP such that the output of one LPASP is the input to the following LPASP. Since the LPASP is laser programmable, each LPASP can perform a different subfunction of the overall task. The input and output ports for the LPASP can be the same input and output ports that are required for a parallel LPASP system, which will be discussed later in section 6.3.2. The GO and DONE control signals in the LPASP can be wired between LPASPs to control the processing of information (see Figure 6.3). One approach is to have each LPASP have its own local memory.

To start the process, the host computer would load all of the required information in the first LPASP's memory and set the GO signal. When the first LPASP has completed its processing, it would set its DONE signal, which would also set the next LPASP's GO signal. The final processing

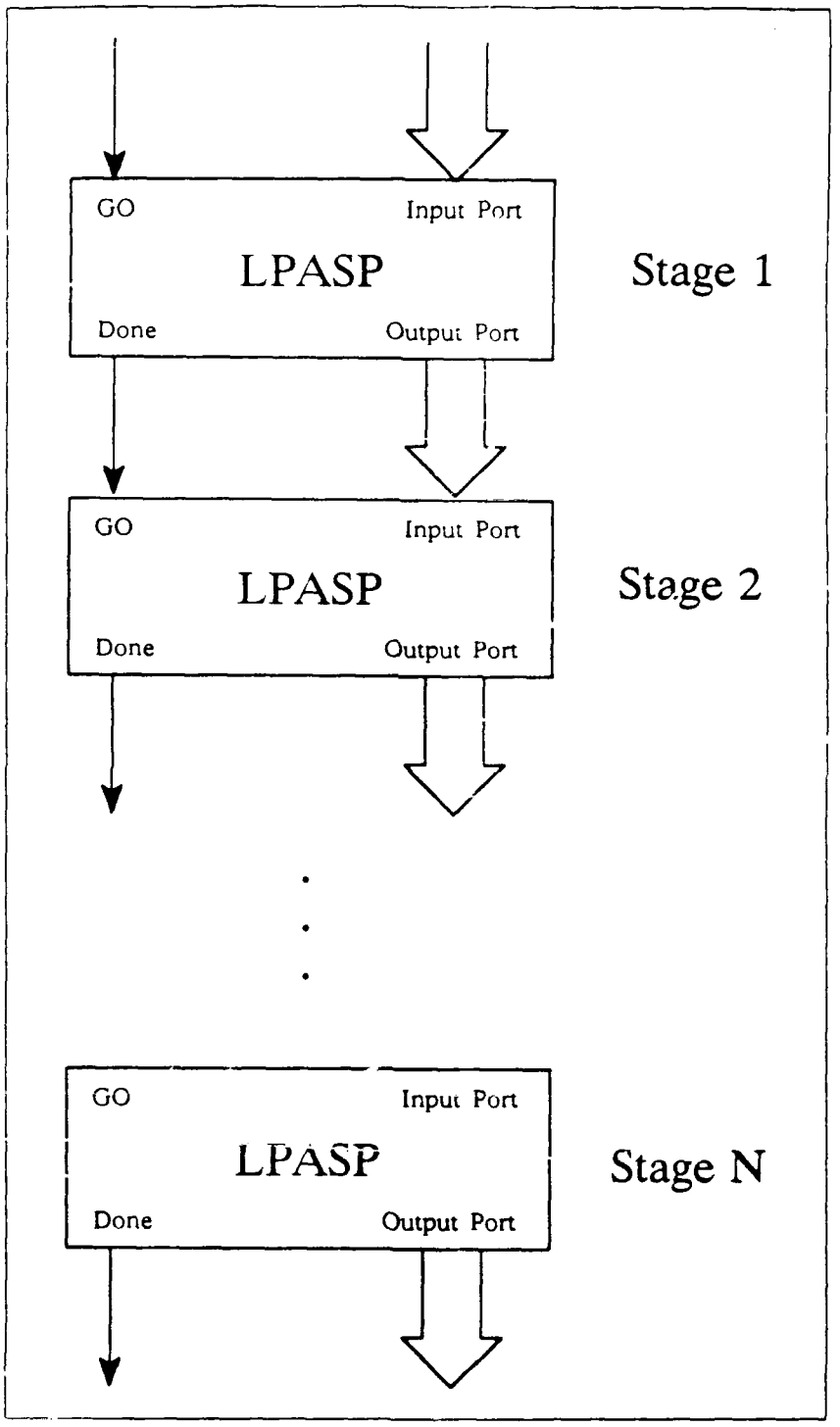

Figure 6.3. Pipelined Architecture

of the first LPASP and the initial processing of the next LPASP would be data transfers using the output port of the first LPASP and the input port of the next LPASP. When the second LPASP completes its processing, it would pass control and data to the next LPASP in the pipeline. This sequence would continue until the last LPASP in the pipeline completed its processing and set its DONE signal, which would flag the host computer of the final result.

If there is a large amount of data transferred between successive LPASPs in the pipeline, then the memory for the LPASP could be placed in a shared configuration. In this shared configuration, an LPASP can read and write data to its own memory and write to the memory of the next LPASP in the pipeline (see Figure 6.4). The LPASP can address one megaword of memory, so addressing would not be a problem. Control logic would have to be used to lock the memory while one of the LPASP was using it to prevent two LPASPs from trying to access memory at the same time. Another solution is to have the memory transfers be synchronized such that every LPASP in the pipeline would write data into its next LPASP's memory at the same time. With the memories in a shared configuration, data busses would not be required between LPASPs.

6.2.4 Application Mapping. Both parallel and pipelined systems must be designed to fit the application that will be used by the system. If the application is partitionable into sequential subfunctions that can be executed concurrently, then the application may be used on a pipelined system. If the application can be divided into independent subprograms that can be executed concurrently on separate hardware, then the application can be used on a parallel system.

The first application for the LPASP is the Kalman filter algorithm. A Kalman filter consists of propagation equations and update equations(17). There have been numerous papers written on how to map the Kalman filter algorithm to a distributed architecture. There are two papers that offer some very good approaches to the problem. The first paper was written for AFWAL by Dr. Neal A. Carlson of Integrity Systems. Dr. Carlson's paper presents a federated filter architecture that combines both parallel processing and pipelining(3). The second paper was written for the Office of Naval Research by Radhakisan S. Baheti and David R. O'Hallaron of the GE Research and Development Center. The second paper mainly addresses mapping a matrix multiplication to a linear array of processors(1).

6.2.4.1 Federated Filter Architecture. The federated filter architecture combines both parallel processing and pipelining into one coordinated system. The processing system assumes a navigation system comprised of a suite of sensors or sensor subsystems with local filtering capabilities, whose outputs are subsequently combined by a larger master filter (see Figure 6.5). Dr.

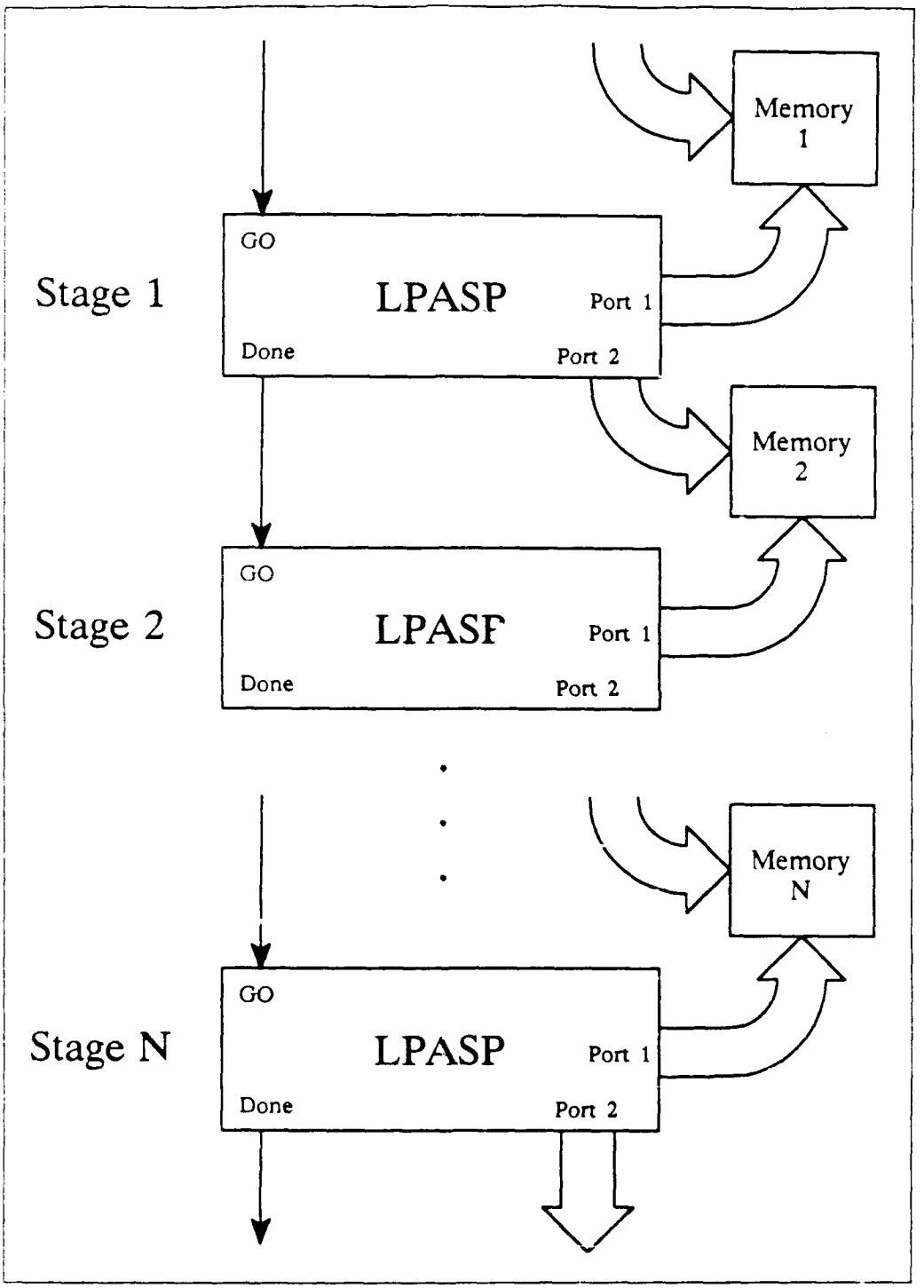

Figure 6.4. Shared Memory Architecture

Carlson's objectives in designing the federated filter was to "synthesize a two-stage alternative to the fully optimal, single-filter architecture that: makes efficient use of local filters for measurement preprocessing prior to master filter processing; incurs little or no loss in performance (estimation accuracy) relative to the theoretically optimal solution; and is stable and robust across the full spectrum of operation conditions to be encountered"(3).

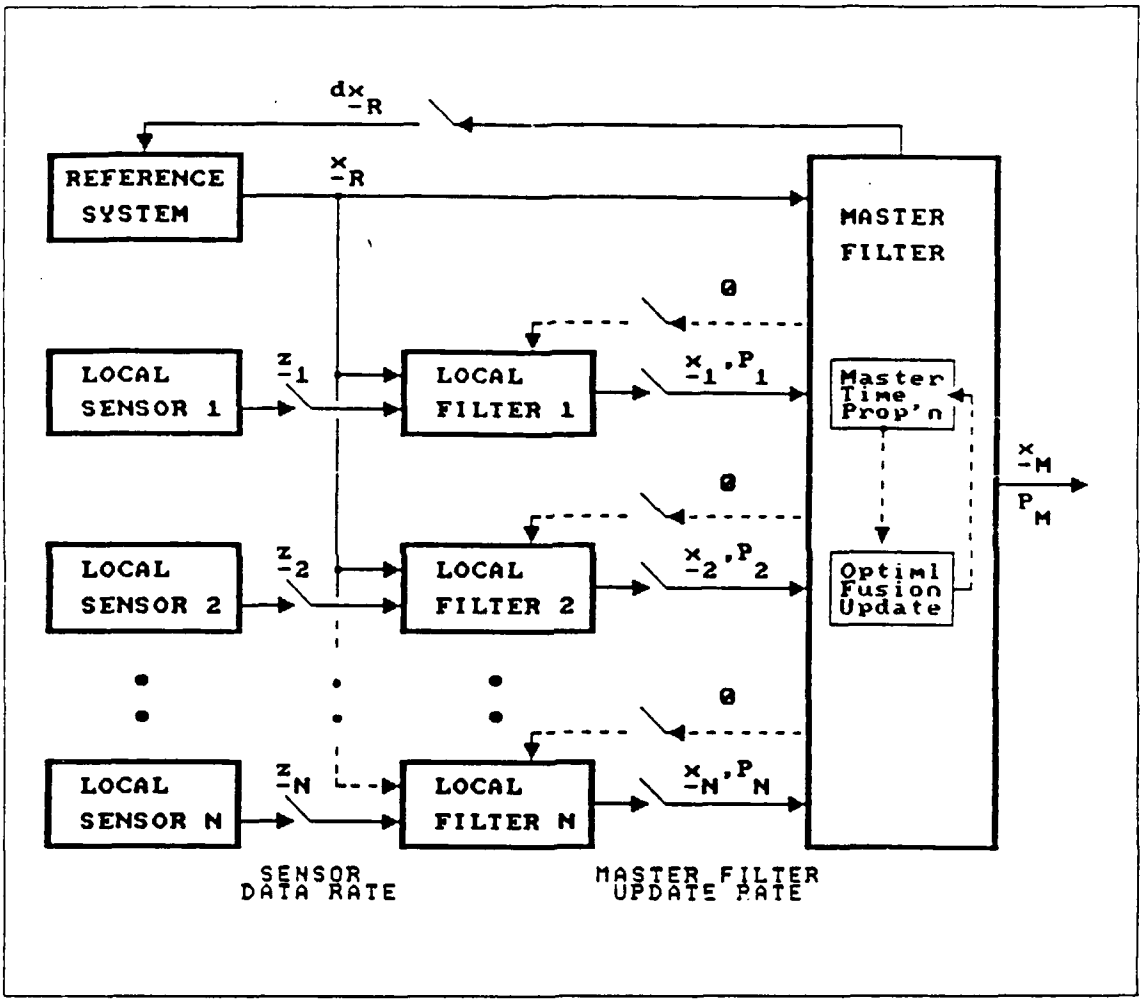

Figure 6.5. Federated Kalman Filter

Each local filter estimates its local state by processing a sequence of measurements during the prefiltering interval. Each local filter state contains all significant elements that are directly observable via these measurements. Each local filter also implements a sufficiently accurate dynamics model for time propagation of the local state and its covariance between measurement times. The results of the local filter are sent to the master filter.

The master filter takes the preprocessed results from all of the local filters and generates the combined (fused) solution. The master filter may or may not feed back its results to the local filters. For optimal accuracy, the master filter must either a) divide and feed back the fused information to the local filters, zeroing its own information, or b) retain the fused information, and zero the local filters' information.

6.2.4.2 Linearly Arrayed Parallel Kalman Filter. The GE Research and Development Center has also been studying ways of mapping the Kalman filter algorithm onto a parallel processing system, but their approach is somewhat different. The researchers at GE concentrated their efforts on reducing the $O(n^3)$ floating point computations required by the Kalman filter algorithm. They accomplished a reduction in computations by performing the vector-matrix multiplications and the matrix-matrix multiplications of an n-state Kalman filter on an $(n+1)$-cell linear array of processors.

The linear array of processors is constructed such that data flows from left to right along an upper communication channel and from right to left along a lower communication channel (see Figure 6.6). Each processor cell, with the exception of the last cell, requires two input ports and two output ports and can communicate with its two neighboring cells. The last processor cell only requires one input port and one output port and can only communicate with the second-to-the-last cell. If this configuration were used for the LPASP, the input data ports would have to be multiplexed into one input port. The same would be true for the output ports.

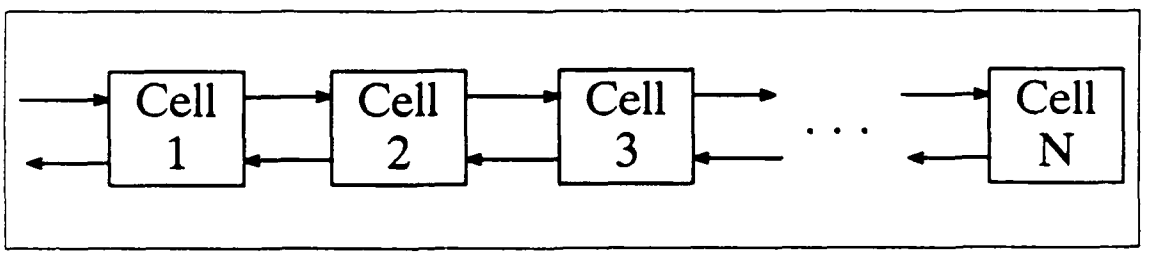

Figure 6.6. Linear Array of Processors

The time update of the covariance matrix for the Kalman filter algorithm is accomplished by updating one column of the covariance matrix on a single computational cell. For an $(n \times n)$ covariance matrix update, n cells are used with an improvement from $O(n^3)$ to $O(n^2)$. The parallel algorithm for the time update is summarized in the following. Before the time update, the ***kth*** cell contains the ***kth*** column of P, the covariance matrix. First, six of the columns of P are shifted to the left and the result is stored in vector q. Next, three columns of P are shifted to the left and the

result is stored in vector r. With the appropriate columns in place, each cell updates its column of P independently(1).

The calculation of the *ith* scalar measurement update of P is slightly different than the time update. Before the *ith* scalar measurement update of P, the *kth* cell contains the *kth* column of P. The elements of H_i, which is the *ith* row of the measurement partial derivative matrix H, are sent from the left to the array of cells. As H_i flows past the *kth* cell, the cell accumulates the *kth* scalar element of H_iP. As scalar elements of H_iP are computed, they are passed back to the left and out of the array(1).

6.3 *Recommendations*

The previous section presented a few of the important options associated with designing a parallel processing system. This section will discuss my recommendations for a parallel processing system. It starts with a suggested approach for parallel processing with the LPASP, followed by the hardware requirements and the software requirements for the LPASP.

6.3.1 Suggested Approach. With the number of options presented in the previous section and the variety of ways that the Kalman filter algorithm can be parallelized, the "best" design for a parallel LPASP system cannot be determined without detailed knowledge of the exact application. The beauty of the LPASP chip is its inherent flexibility, such that it can be programmed to perform any number of different algorithms in either a parallel architecture or a pipelined architecture.

The major design issue, with respect to the LPASP, is whether the LPASP contains the necessary hardware to accommodate all of the various parallel configurations. More specifically, the LPASP must support the processor-to-memory configuration, which requires a bi-directional port and appropriate control lines, and it must support the PE-to-PE configuration, which requires an input port, an output port, an external memory port, and appropriate control lines. If the LPASP can support both of these communication configurations then it can be placed in any parallel or pipeline architecture and can connect to any interconnection network.

There is one other point that should be emphasized. The internal architecture of the LPASP is optimized for the dot product routine that is used for vector-vector, matrix-vector, and matrix-matrix multiplication(4). There are a number of algorithms for performing a parallel matrix-matrix multiplication(7, 2, 10, 13). When a parallel architecture is designed, it should take advantage of the optimization by storing an entire column of the matrix in each of the LPASPs. By broadcasting the elements of the row of the other matrix, a matrix-matrix multiplication can be reduced

from an $O(n^3)$ computation to an $O(n^2)$ computation and still take advantage of the dot product routine(13).

6.3.2 Hardware Requirements. The architecture for the LPASP has been designed, and it would be best to minimize the modifications made to the architecture to accommodate parallel processing. Parallel processing requires input ports, output ports, and control lines for interprocessor and external memory communications. Each port requires 32 pins for integer transfers or 64 pins for floating point transfers. It would take 192 pins just to satisfy the ports for floating point transfers. This number does not include the 40 pins needed for memory addressing and all of the other pins required for miscellaneous control signals. Unfortunately, packages with pin counts in excess of 240 pins are not available from MOSIS.

The LPASP already has two 32-bit independently addressable external memory ports (see Figure 4.1). The two ports can be coordinated to read or write 64-bit floating point numbers, or they can work separately to access two different 32-bit integer numbers. Each memory port has a 20-bit address associated with it. One solution to the communication port problem is to use one of the memory ports as a memory/input port and the other as a memory/output port. The high order bits of the address could be used to choose between its memory or one of its neighboring processors. For example, if the three highest order bits were used for communication addressing, then each processor could access 128K of memory using address 000 and communicate with seven other processors using addresses 001 through 111. If a hypercube interconnection network was used then each processor would have seven nearest neighbors, and a parallel system could be constructed with 2^7 or 128 processing elements.

6.3.3 Software Requirements. There is one minor problem with the solution of using the memory ports as input and output ports, but this problem can be solved by the software. The memory ports are only 32 bits wide and can not handle 64-bit floating point numbers. The solution is to use two 32-bit transfers to pass 64-bit floating point numbers between processors. General purpose registers R1 and R2 can be used in addition to the MBR to store the data off of the data pads during a data transfer (see Figure 3.2). Though the floating point transfers will take two clock cycles, this should only impact the dot product routine by at most one clock cycle because the data transfer can occur while the floating point multiplier and the floating point adder are processing. The additional clock cycle may be needed to transfer the high order bits of the floating point number from the lower datapath to the upper datapath.

The recommended configuration is to use the lower datapath as the input port and the upper datapath as the output port. The reason for this selection is that it is natural to send the high order bits first followed by the low order bits. For a data input, the high order bits from another processor are sent to the lower datapath input port in the first clock cycle. Then in the second clock cycle those high order bits can be shifted to the upper datapath while the low order bits from another processor are being sent to the lower datapath input port. Likewise, for a data output, in the first clock cycle the low order bits can be shifted to the upper datapath while the high order bits are being sent from the upper datapath output port to another processor. Then in the second clock cycle the low order bits can be sent from the upper datapath output port to another processor. With separate ports, data can be sent and received in the same clock cycle.

6.4 Conclusion

Before a parallel processing system can be designed, some fundamental issues must be resolved. The four design decisions associated with the processor communication are the operating mode, the control strategy, the switching methodology, and the network topology. A decision must be made between using a centralized memory architecture or a decentralized memory architecture. There is also the issue of whether to go with a parallel architecture or a pipelined architecture. Most of these issues will depend on the application that is being mapped to the system. The first application of the LPASP is the Kalman filter algorithm. Two parallel versions of the Kalman filter algorithm have been presented.

The LPASP is a very flexible chip that can be used in a variety of parallel architectures without modifying the architecture of the chip. The LPASP contains two external memory ports that can be used as an input port and an output port. The two ports allow the LPASP to be connected to any interconnection network for maximum flexibility. The "best" configuration and network that should be used for a Kalman filter application depends on the algorithm that is used, the number of sensors available in the navigation system, and the number of states in the Kalman filter matrix.

VII. Conclusions and Recommendations

7.1 Introduction

The original goal of this thesis was to design the double precision floating point adder, integrate it with the other macrocells into an LPASP chip, and test the chip. The chip was to be the largest and fastest chip ever designed at AFIT. Due to complications with the original LPROM and the floating point multiplier, the ultimate goal of creating the LPASP chip could not be accomplished at this time. As a result, the direction of this thesis was modified to include the analysis and design of a new LPROM and the investigation of parallel applications of the LPASP.

The objectives of this thesis effort have been accomplished with the achievement of some significant contributions. The remaining sections will review the contributions made by this thesis, discuss some of the lessons learned, and make some recommendations for future work.

7.2 Contributions

Five chips were created by this thesis effort. Of the five chips, four of them worked and one of them worked for the majority, but not all, of the possible stimulus cases. The following paragraphs will present the contributions made by each of the chips.

The research effort toward the floating point (FP) adder lead to the creation of the optimized carry multiplexed (OCM) adder architecture, the most efficient VLSI adder design to date. The 57-bit OCM adder test chip verified the architecture, design, and implementation of the 58-bit adder used as the heart of the FP adder. The timing analysis for the design of the OCM adder cells was used as the basis for the transistor sizing in the remaining substructures of the FP adder. The OCM test chip also contributed to the realization of the limitations of using on-chip probes for AC parametric testing.

The FP adder chip was the most complex and demanding of all of the chips that were created. Though the chip does not work for all of its possible inputs, it does work for the vast majority of the input cases and can be considered a major achievement. The contributions of the FP adder are not only in its completion, but also in the unique implementations of some of its substructures. The first unique implementation was the creation of a master-slave flip-flop that combined N and P pass transistors in such a way as to decrease the number of required control lines and cut in half its original size. The second unique implementation was the creation of an arithmetic linear shifter using N pass transistors to decrease its original size by one-third. Without the size reduction in the two linear shifters, the 2300λ size restriction on the FP adder could not have been achieved. A

third unique implementation was the creation of the priority encoder, which used pseudo-NMOS NOR gates and muxes to create a relatively small structure.

The two laser technology test chips gave proof of concept to the application of diffusion linking and metal cutting with a laser. They also provided knowledge and understanding of the process parameters and the limitations of diffusion linking and laser scribing. The two chips gave insight into the design rules that were used to create the new LPROM chip.

The LPROM chip was the most successful chip that was created. The density of the LPROM chip is twice that of the original LPROM. The chips substructures were logically correct and the laser programming yield was 100% for all four of the test chips. The read access time is very low and demonstrates the feasibility of using a metal cutting LPROM in the LPASP.

7.3 Lessons Learned

There are four significant lessons that were learned from the work that was performed in accomplishing the goals of this thesis. The first lesson was the importance of timing analysis. When the design for the cells of the adder was first started, everything was analyzed, but the layout effort got behind schedule. As a result, the timing analysis from the adder cells was used to create the majority of the cells that were designed for the FP adder. A better approach would be to alternate between analysis and layout throughout the entire project. If this approach would have been followed, the FP adder may have had a smaller propagation delay time because the multiplexer control drivers would have been more appropriately sized.

The second lesson was the use of latching data on-chip when timing considerations are significant but pins are not plentiful. Probe pads are great for gaining visibility of the logic within the chips, but they cannot be used very effectively to provide timing information because of the capacitance of the probes. A better method is to use latches that are controlled by an external clock. After the data is latched, its logic can either be probed or serially shifted to an output pin.

A third lesson that was learned is not to be so concerned about always reducing size. There were a few occasions where something was designed extremely small and then had to be redesigned larger to fit into the data path pitch. The best approach is to determine an appropriate data path size and then keep that size throughout all of the substructures. By keeping a consistent data path size, the power and ground lines of the substructures will route easily and a standard cell library can be produced.

The fourth lesson that was learned was to maximize the use of labels. Labels are the best way to document a design. They help to maintain continuity between substructures and help to detect errors in the debugging process. A major reason for the success of the five chips is the extensive use of labels in the design. Labels can get in the way during ESIM testing, but a quick and easy way to get around the ESIM problem was found. The solution requires three steps. The first step is to create a macro that selects a cell, edits the cell, and erases labels. The second step is to expand the chip on MAGIC with only the "labels" layer exposed. The final step is to repeatedly point at clusters of labels and press the macro key. Eventually, every cell's label layer will be deleted and the screen will be blank. If key labels are placed on the layout then the chip can be CIFed with only the key labels.

7.4 *Future Work*

There are four areas of work that should be accomplished before the LPASP chip is ready for fabrication. The first and most important task is to layout the floating point multiplier and create a test chip of it. The second task is to analyze the multiplexer driver logic of the FP adder and properly size the drivers. The third task is to create test chips of the integer data path of the LPASP to verify the logic that was designed by Capt Comtois. The fourth task is to start with the parallel applications foundation that was presented in this thesis and create a specific parallel application for the LPASP. This last task is optional, depending on the purpose of the first version of the LPASP.

Appendix A. *Pin Out for OMC Adder Test Chip*

Pin #	Description	Pin #	Description
1	A_0, A_{22}, A_{29}, A_{50}	21	A_8, A_{15}, A_{36}, A_{43}
2	A_1, A_{23}, A_{30}, A_{51}	22	A_9, A_{16}, A_{37}, A_{44}
3	A_2, A_{24}, A_{31}, A_{52}	23	A_{10}, A_{17}, A_{38}, A_{45}
4	A_3, A_{25}, A_{32}, A_{53}	24	A_{11}, A_{18}, A_{39}, A_{46}
5	GND	25	GND
6	A_4, A_{26}, A_{33}, A_{54}	26	A_{12}, A_{19}, A_{40}, A_{47}
7	Load bits 43 56	27	A_{13}, A_{20}, A_{41}, A_{48}
8	A_5, A_{27}, A_{34}, A_{55}	28	A_{14}, A_{21}, A_{42}, A_{49}
9	A_6, A_{28}, A_{35}, A_{56}	29	Load bits 15 28
10	B_0, B_{22}, B_{29}, B_{50}	30	NC
11	B_1, B_{23}, B_{30}, B_{51}	31	Load bits 29 42
12	B_2, B_{24}, B_{31}, B_{52}	32	GO
13	Load bits 0 14	33	B_8, B_{15}, B_{36}, B_{43}
14	B_3, B_{25}, B_{32}, B_{53}	34	B_9, B_{16}, B_{37}, B_{44}
15	Vdd	35	Vdd
16	B_4, B_{26}, B_{33}, B_{54}	36	B_{10}, B_{17}, B_{38}, B_{45}
17	B_5, B_{27}, B_{34}, B_{55}	37	B_{11}, B_{18}, B_{39}, B_{46}
18	B_6, B_{28}, B_{35}, B_{56}	38	B_{12}, B_{19}, B_{40}, B_{47}
19	A_7	39	B_{13}, B_{20}, B_{41}, B_{48}
20	B_7	40	B_{14}, B_{21}, B_{42}, B_{49}

Appendix B. *Pin Out for Floating Point Adder Test Chip*

Pin #	Description	Pin #	Description	Pin #	Description	Pin #	Description
1	Vdd	22	GND	43	Vdd	64	GND
2	I/O_0	23	I/O_{16}	44	I/O_{42}	65	I/O_{57}
3	I/O_1	24	I/O_{17}	45	I/O_{43}	66	I/O_{58}
4	I/O_2	25	I/O_{18}	46	I/O_{44}	67	I/O_{59}
5	I/O_3	26	I/O_{19}	47	I/O_{45}	68	I/O_{60}
6	I/O_4	27	I/O_{20}	48	Load A	69	I/O_{61}
7	I/O_5	28	I/O_{21}	49	PQ2	70	I/O_{62}
8	I/O_6	29	Vdd	50	GND	71	Vdd
9	I/O_7	30	I/O_{22}	51	Enable	72	I/O_{63}
10	GND(SC)	31	I/O_{23}	52	I/O_{46}	73	NC
11	I/O_8	32	I/O_{32}	53	I/O_{47}	74	NC
12	I/O_9	33	I/O_{33}	54	I/O_{48}	75	NC
13	I/O_{10}	34	I/O_{34}	55	I/O_{49}	76	I/O_{31}
14	I/O_{11}	35	I/O_{35}	56	I/O_{50}	77	I/O_{30}
15	Vdd	36	GND	57	Vdd	78	GND
16	PQ1	37	I/O_{36}	58	I/O_{51}	79	I/O_{29}
17	Load B	38	I/O_{37}	59	I/O_{52}	80	I/O_{28}
18	I/O_{12}	39	I/O_{38}	60	I/O_{53}	81	I/O_{27}
19	I/O_{13}	40	I/O_{39}	61	I/O_{54}	82	I/O_{26}
20	I/O_{14}	41	I/O_{40}	62	I/O_{55}	83	I/O_{25}
21	I/O_{15}	42	I/O_{41}	63	I/O_{56}	84	I/O_{24}

Appendix C. *Pin Out for LPROM Test Chip*

Pin #	Description	Pin #	Description
1	$Data_0$	21	NC
2	$Data_1$	22	NC
3	$Data_2$	23	NC
4	$Data_3$	24	$Addr_0$
5	GND	25	GND
6	$Data_4$	26	NC
7	$Data_5$	27	NC
8	$Data_6$	28	$Addr_2$
9	$Data_7$	29	$Addr_3$
10	$Data_8$	30	$Addr_4$
11	$Data_9$	31	$Addr_5$
12	Precharge	32	$Addr_6$
13	$Addr_1$	33	$Addr_7$
14	$Data_{10}$	34	NC
15	Vdd	35	Vdd
16	$Data_{11}$	36	NC
17	$Data_{12}$	37	NC
18	$Data_{13}$	38	NC
19	$Data_{14}$	39	NC
20	$Data_{15}$	40	NC

Bibliography

1. Radhakisan S. Baheti and David R. O'Hallaron. Efficient parallel implementation of target tracking kalman filter. In *Proceedings of the 27th IEEE Conference on Decision and Control*, pages 376-381. The Institute of Electrical and Electronic Engineers, Inc., New York, NY 10017, 1988.

2. D. W. Blevins et al. Blitzen - a highly integrated massively parallel machine. In *Proceedings of the 2nd Symposium on the Frontiers of Massively Parallel Computation*. The Symposium on the Frontiers of Massively Parallel Computation, Fairfax VA, 1988.

3. Neal A. Carlson and Clark M. Neily. Distributed kalman filter architecture. Technical Report TR-87-001, Integrity Systems, Wincester, MA 01890, June 1987.

4. John H. Comtois. Architecture and design for a laser programmable double precision floating point application specific processor. Master's thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1988.

5. Nathan Davis. Class handout. Distributed in EENG 688, Intermediate Computer Architecture, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1989.

6. Joe DeGroat. Personal interview. WPAFB, OH, November 1988.

7. Dwayne C. Ethridge. Current research in parallel microprocessing systems at los alamos Technical Report LA-UR-84-1324, Los Alamos National Laboratory, Los Alamos, NM 87545, May 1984.

8. Erik J. Fretheim. Internal communication on the double precision multiplier. Technical report, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, June 1987.

9. David M. Gallagher. Rapid prototyping of application specific processors. Master's thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987.

10. Peter Y. Hsu. A parallel vlsi architecture for sparse matrix computation. Master's thesis, School of Engineering, University of Illinios, Urbana IL, May 1982.

11. J. J. Lee and N. R. Strader. Cmos rom arrays programmable by laser beam scanning. *IEEE Journal of Solid State Circuits*, SC-22(4), August 1987.

12. Richard W. Linderman et al Design and application of an optimized xrom silicon compiler. In *Paper submitted to the IEEE Journal on Computer Aided Design for Integrated Circuits*. The Institute of Electrical and Electronic Engineers, Inc., New York, NY 10017, 1988.

13. Y. L. C. Ling et al. A vlsi robotics vector processor for real-time control. In *Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, pages 303 308. The Institute of Electrical and Electronic Engineers, Inc., New York, NY 10017, 1988.

14. Matthew Rhodes. Lincoln site report summary. Technical report, Massachusetts Institute of Technology Lincoln Laboratory, Lexington Massachusetts, March 1986.

15. Matthew Rhodes. Personal interview. Lincoln Laboratories, Hanscom AFB, MA, March 1989.

16. Michael W. Scriber and Chuck Wardin. Double precision floating point adder. Technical report, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1989.

17. Joseph K. Solomon. Development of the extended kalman filter for the advanced completely integrated reference instrumentation system (ciris). Master's thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1989.

18. Craig S. Spanburg. Laser programming integrated circuits. Master's thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987.

19. John J. Tillie. A diffusion link laser programmable read only memory and automated programming station. Master's thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1988.

20. Hiroshi Yamaguchi and others. Laser cutting of aluminum stripes for debugging integrated circuits. *IEEE Journal of Solid State Circuits*, SC-20(6), December 1985.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/CCE/ENG/89D-6	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, OH 45433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION CIRIS Section 6585th Test Group 8b. OFFICE SYMBOL (If applicable) CDN	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Holloman AFB, NM 88330	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Technology Development and Circuit Design for a Parallel Laser Programmable Floating Point Application Specific Processor (UNCLASSIFIED)

12. PERSONAL AUTHOR(S)
Michael W. Scriber, B.S., Captain, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Thesis	FROM _____ TO _____	1989 December	110

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
12	06		Computer Architecture, Floating Point Adder, Laser Programming, Parallel Processing, VISI Design, Integrated Circuits, VHSIC, ROM
09	01		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Thesis Chairman: Keith R. Jones, Captain, USAF
Instructor of Electrical Engineering

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Keith Jones, Captain, USAF	513-255-3576	AFIT/ENG

DD Form 1473, JUN 86 *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19 Abstract

The laser programmable floating point application specific processor (LPASP) is a new approach at rapid development of custom VLSI chips. The LPASP is a generic application specific processor that can be programmed to perform a specific function. The effort of this thesis is to develop and test the double precision floating point adder and the laser programmable read-only memory (LPROM) that are macrocells within the LPASP. In addition, the thesis analyzes the applicability of an LPASP parallel processing system.

The double precision floating point adder is an adder/subtractor macrocell designed to comply with the IEEE double precision floating point standard. An 84-pin chip of the adder was fabricated using 2 micron feature sizes. The fastest processing time was measured at 120 nanoseconds over 23 worst case test vectors. The adder uses the optimized carry multiplexed (OCM) adder that was developed at AFIT.

The OCM adder is a new adder architecture that uses four parallel carry paths to attain a performance time on the order of $O(\sqrt[3]{n})$ with a gate count on the order of $O(n)$. The redundant logic associated with the parallel propagation banks is eliminated in the OCM adder so that the largest bit-slice of the adder contains only eight 2-to-1 multiplexer gates. A 57-bit adder was fabricated using 2 micron feature sizes. The processing time for the adder is 31 nsec.

The laser programmable read-only memory (LPROM) is programmed by using an argon-ion laser to cut transistor links. The LPROM was designed to provide a post-fabrication programmable capability to a MOSIS compatible ROM. A 256 by 16 bit LPROM was fabricated using 2 micron feature sizes. The chips were laser programmed with a laser programming yield of 100% and an off-chip read access time of 23 nanoseconds.

Vita

Captain Michael W. Scriber [redacted] in 1981 [redacted] attended Rensselaer Polytechnic Institute, where he graduated Magna Cum Laude with a Bachelor of Science degree in Computer and Systems Engineering in May 1985. Upon graduation, he received a commission in the USAF through the ROTC program. He was employed as a computer engineer for Infosphere, Portland, Oregon, until called to active duty in October 1985. His first assignment was to the 6595th Shuttle Test Group, Vandenberg AFB, as a Space Shuttle computer software engineer until November 1987. He then served as a Titan II booster electrical engineer in the 6595th Aerospace Test Group, Vandenberg AFB, until entering the School of Engineering, Air Force Institute of Technology, in May 1988. Capt Scriber's assignment after graduation is to Space Systems Division, Los Angeles AFB, to work on the Star Lab project as a computer project engineer for the Space Defense Experiments SPO.

Permanent address: 17926 SE Addie Avenue
Milwaukie, OR 97267